PATENT ABSTRACTS OF JAPAN

(11)Publication number : **10-091443**

(43)Date of publication of application : **10.04.1998**

(51)Int.CI. G06F 9/42

G06F 9/30

G06F 9/46

(21)Application number : **09-135923**

(22)Date of filing : **08.05.1997**

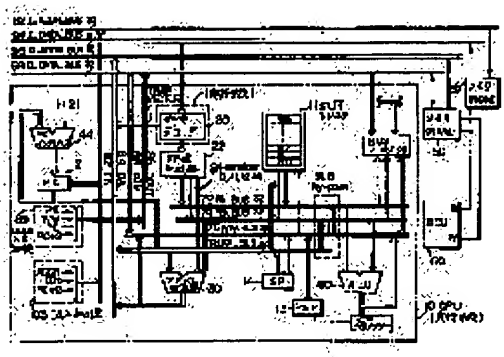
(71)Applicant : **SEIKO EPSON CORP**

(72)Inventor : **KUBOTA SATORU**
KUDO MAKOTO
MIYAYAMA YOSHIYUKI
SATO HISAO

(30)Priority

Priority number : **08127541** Priority date : **22.05.1996** Priority country : **JP**

(54) INFORMATION PROCESSING CIRCUIT, MICROCOMPUTER AND ELECTRONIC EQUIPMENT



(57)Abstract:
PROBLEM TO BE SOLVED: To efficiently describe a processing for dealing with a stack pointer with short instruction length, to efficiently describe the processings of execution, register saving and register restoration and to improve the processing speed of an interruption processing and sub-routine call return.
SOLUTION: CPU 10 decodes a stack pointer-only instruction group which contains a register SP14 only for the stack pointer and sets SP14 to be a mute operand by an instruction decoder 20. The stack pointer-only instruction group is executed in terms of hardware by using a generalpurpose register 11, PC12, SP14, an address adder 30, ALU 40 a PC incrementor 44, internal buses 72, 74, 76 and 78, internal signal lines 82, 84, 86 and 88 and external buses 92, 94, 96 and 98. The stack pointer-only instruction group contains an SP relative load instruction, a stack pointer shift instruction, a call instruction, a return instruction, a continuous push instruction and a continuous pop instruction.

LEGAL STATUS

[Date of request for examination]	12.12.2002
[Date of sending the examiner's decision of rejection]	01.06.2004
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]	withdrawal
[Date of final disposal for application]	20.07.2004
[Patent number]	
[Date of registration]	
[Number of appeal against examiner's decision of rejection]	
[Date of requesting appeal against examiner's decision of rejection]	
[Date of extinction of right]	

CLAIMS

[Claim(s)]
[Claim 1] The information-processing circuit characterized by to have the object code which makes an implied operand the stack-pointer dedicated register only used for stack pointers, and this stack-pointer dedicated register, to decode the object code of the instruction group only for stack pointers the processing based on this stack-pointer dedicated register was described to be, and to include a decode means output a control signal based on this object code, and an activation means perform said instruction group only for stack pointers based on the contents of said control signal and said stack-pointer dedicated register.
[Claim 2] In claim 1, said instruction group only for stack pointers includes the load instruction which has transfer register specific information in object code. When said decode means decodes said load instruction and said activation means executes said load instruction, Either [at least] the data transfer from the first area to the first given given register on memory, or the data transfer from said first given register to said first given area The information processing circuit characterized by carrying out based on the register address specified by the memory address specified with said stack-pointer dedicated register, and said transfer register specific information.
[Claim 3] The information processing circuit where the offset information said whose load instruction is the information about the offset for specifying the address of said first area on said memory is included in object code in claim 2, and said activation means is characterized by specifying said memory address using the contents and said offset information on said stack-pointer dedicated register.
[Claim 4] In claim 3, said offset information includes the immediate offset information given by the immediate, and the data size information about the size of the given data on memory. Said activation means is based on said immediate offset information and said data size information. The information processing circuit which carries out the left logic shift of said immediate offset information, and is characterized by specifying said memory address with the value which generated the offset value and added the contents and said offset value of said stack-pointer dedicated register.
[Claim 5] The information-processing circuit characterized by to change the contents of said stack-pointer dedicated register based on said migration information in case said decode means decodes said stack-pointer migration instruction and said activation means executes said stack-pointer migration instruction including a stack-pointer migration instruction for said instruction group only for stack pointers to have migration information in object code, and move a stack pointer to it in either claim 1 - claim 4.
[Claim 6] The information processing circuit where said instruction-execution means is

characterized by processing at least one side of the processing which subtracts said immediate migration information from the contents of the processing adding said immediate migration information and contents of said stack-pointer dedicated register, and said stack-pointer dedicated register in claim 5 including the immediate migration information that said migration information was given by the immediate.

[Claim 7] In either claim 1 - claim 6, two or more registers which were able to be continuously set in order are included. Said instruction group only for stack pointers includes either [at least] the continuation push instruction which has two or more register specific information in object code, or a continuation pop instruction. Said decode means decodes either [at least] said continuation push instruction or a continuation pop instruction. When said instruction-execution means executes either [at least] said continuation push instruction or said continuation pop instruction, At least one side of the processing which carries out multiple-times pop succeeding said two or more registers from the processing which carries out a multiple-times push from said two or more registers continuously to the stack prepared in memory, and said stack The information processing circuit characterized by attaining to the memory address specified according to the contents of said stack-pointer dedicated register, and carrying out based on said two or more register specific information.

[Claim 8] In claim 7, n general-purpose registers specified by the register numbers from 0 to n-1 are included. One [at least] object code of said continuation push instruction and a continuation pop instruction as said two or more register specific information The last register number as which either of said register numbers was specified is included. At least one side of the processing which carries out multiple-times pop succeeding said two or more registers from the processing in which said activation means carries out a multiple-times push continuously to the stack prepared in memory from two or more registers to the register specified by said last register number from a register 0, and said stack The information processing circuit characterized by carrying out based on the memory address specified according to the contents of said stack-pointer dedicated register.

[Claim 9] In claim 7 or either of 8 said activation means the contents of the either given register of two or more of said registers The write-in means written in the stack prepared in memory based on the memory address specified with said stack-pointer dedicated register, A count count means of writing to count the count of writing to said stack by said write-in means, A comparison means to compare the value of said count of writing counted with said count means and said two or more register specific information is included. A write-in memory address generation means by which said write-in means generates the write-in memory address for adding the first input and second input with an adder, and specifying a writing place, It controls so that the first input of said adder serves as the contents of the stack-pointer dedicated register at the time of activation initiation of the instruction only for continuation. The first [which was generated by the write-in address-generation means after it] input-control means controlled to write in and to become the address, The second input-control means which outputs the offset value when writing 1 word in the second input of said adder from said stack, The contents of the register specified with the value which subtracted said count of writing from said two or more register specific information The information processing circuit characterized by controlling two or more writing to said stack from a register and write-in termination based on the comparison result of said comparison means including the means written in said stack based on said write-in memory address.

[Claim 10] The contents of the stack by which said instruction-execution means was formed in

memory in either of claims 7-9 based on the memory address specified with said stack-pointer dedicated register are read. A read-out means to store in the either given register of two or more of said registers, A count count means of read-out to count the count of read-out from said stack by said read-out means, A comparison means to compare the value of said count of read-out counted with said count means and said two or more register specific information is included. A write-in memory address generation means by which said read-out means generates the write-in memory address for adding the first input and second input with an adder, and specifying a writing place, It controls so that the first input of said adder serves as the contents of the stack-pointer dedicated register at the time of activation initiation of the instruction only for continuation. The first [which was generated by the read-out address-generation means after it] input-control means controlled to read and to become the address, The second input-control means which outputs the offset value when writing 1 word in the second input of said adder from said stack, The contents of said stack are read based on said read-out memory address. The information processing circuit characterized by controlling read-out and read-out termination of the contents of said stack based on the comparison result of said comparison means including a read-out means to store in the register specified based on said count of writing.

[Claim 11] In either claim 1 - claim 10, the program counter register only for program counters is included. The branch instruction said whose instruction groups only for stack pointers are the instruction which branches to a subroutine, and a return instruction from said subroutine is included. When said decode means decodes said branch instruction and said instruction-execution means executes said branch instruction, At least one side of the return to shunting of the contents of said program counter register to the second given area of the stack prepared in memory, and the program counter register of the contents of said second area The information processing circuit characterized by including the means performed based on the memory address specified with said stack-pointer dedicated register, and a means to update the contents of said stack-pointer dedicated register based on said shunting and said return.

[Claim 12] It is an information processing circuit containing the stack pointer assigned to two or more registers which were able to be continuously set in order, and one of general-purpose registers. The object code of one [at least] instruction of the continuation push instruction which has two or more register specific information in object code, and a continuation pop instruction is decoded. When at least one side of a means to output a control signal based on this object code, and said continuation push instruction and said continuation pop instruction is performed, At least one side of the processing which carries out multiple-times pop succeeding said two or more registers from the processing which carries out a multiple-times push from said two or more registers continuously to the stack prepared in memory, and said stack The information processing circuit characterized by including the means performed based on the memory address specified according to the contents of said control signal and said stack-pointer dedicated register, and said two or more register specific information.

[Claim 13] The information processing circuit characterized by being a RISC method in either claim 1 - claim 12.

[Claim 14] The information processing circuit characterized by decoding a fixed-length instruction and performing executive operation in either claim 1 - claim 13 based on this instruction.

[Claim 15] The microcomputer characterized by including a means to perform I/O with an information processing circuit, a storage means, and the exterior according to claim 1 to 14.

[Claim 16] The microcomputer characterized by performing the program of the language which

has the structure where relate with said stack pointer and the storage region of an auto variable is secured in claim 15.
[Claim 17] Electronic equipment characterized by including the microcomputer indicated by claim 15 or either of 16.

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention] This invention relates to the electronic equipment constituted using the microcomputer which contains an information processing circuit and said information processing circuit, and this microcomputer.
[0002]
Background Art and Problem(s) to be Solved by the Invention] Conventionally, in the microcomputer of a RISC method which can process 32-bit data, the fixed-length instruction of 32-bit width of face was used. The reason is that the time amount which decoding of an instruction takes can be shortened compared with the case where a variable-length instruction is used, and it can make the circuit scale of a microcomputer small if a fixed-length instruction is used.
[0003] However, also in a 32-bit microcomputer, there are also many instructions which are not needed especially 32 bits. Therefore, if 32 bits describes all instructions, the instructions which a redundant part produces in an instruction will increase in number, and the utilization ratio of memory will worsen.
[0004] then, this invention -- the person was performing examination about the microcomputer which processes the fixed-length instruction of bit width of face shorter than the bit width of face of the data which can be processed in order to raise the utilization ratio of memory, without complicating a control circuit.
[0005] However, if 32 bit fixed length's instruction is only made into a 16-bit fixed length, for example, the following troubles will arise.
[0006] That is, in the microcomputer of a RISC method, in order to think the versatility of processing and an instruction set as important, when processing which deals with a stack pointer was performed, said processing was performed using the instruction which makes a general-purpose register applicable to actuation, using one of general-purpose registers as a stack pointer. Therefore, assignment of the general-purpose register currently used as a stack pointer is needed for that of said instruction at the time of describing such processing.
[0007] For example, when the instruction which makes a general-purpose register applicable to actuation describes the processing which transmits the data specified by the memory address which applied the given offset value to the stack pointer to a given register, said offset value, the information which specifies said given register, and the information which specifies the register currently used as a stack pointer are needed for the object code of the instruction.
[0008] Thus, since the information specified by object code increases when describing the processing which deals with a stack pointer with the instruction which makes a general-purpose register applicable to actuation, it becomes difficult to describe the contents of the instruction by the 16-bit fixed length. Also especially into an instruction, if an instruction length is made into 32 bits here, since an instruction also has many instructions which are not needed 32 bits, the

instructions which produce a redundant part will increase in number, and aggravation of the utilization ratio of memory will be invited to an instruction.
[0009] Moreover, since the memory which stores it is also too much needed if an instruction length becomes long, it is desirable that not only a fixed-length instruction but an instruction length can be shortened from a viewpoint of efficient use of memory.
[0010] Moreover, since the instructions which deal with a stack pointer increase in number when performing the program described in the language in which it relates with a stack pointer, for example like C, and the storage region of an auto variable is secured, it is desirable to describe efficiently the instruction which deals with a stack pointer, and to perform it.
[0011] Then, when processing which deals with a stack pointer was performed, by the shortest possible instruction length, the contents of an instruction were described and architecture which can be performed was desired.
[0012] Moreover, especially CPU of a RISC method came to have many general-purpose registers in the interior in order [latest] to raise the engine performance. It is because many processings are made to a high speed inside CPU, without accessing memory by having many registers in the interior. Thus, if it has many internal registers, the number of registers which should be evacuated will increase in the case of interruption processing or processing of the register evacuation at the time of a subroutine call, and restoration.
[0013] The conventional example is explained taking the case of shunting of the register which uses it abundantly in case going into a subroutine also in a stack system instruction **** is sufficient hereafter and it carries out, and return instruction.
[0014] Usually, the instruction set of a microcomputer shunts to the stack which prepared the register of CPU in memory, or has the instruction for restoring. There is a thing with an instruction of dedication for that or a thing with a register indirect-addressing instruction.
[0015] As a technique about the instruction for [said] shunting or restoring, "80386 Pro gramine" (John H.Crawford Patric P.Gelsigner work Iwatani ** translation) has the following description about 80386 of Intel.
[0016] That is, it is an instruction which writes a register in a stack. There are push, pusha, and pushad and there are pop, popa, and popad as an instruction for returning data to a register from a stack.
[0017] When writing a register in a stack with a push instruction push EAX A register is specified as an operand like. This is the case of the 32-bit register EAX. It is push when writing Registers EAX, ECX, EDX, and EBX to a stack altogether. EAXpush ECXpush EDXpush A push instruction is repeated like EBX.
[0018] Thus, if it is operating one register at a time with push and a pop instruction, the size of object code will become large, and a program execution step will also increase, therefore program execution time amount and processing actuation will become slower.
[0019] then, all eight general-purpose registers that 80386 has -- a stack -- writing in -- pusha or -- pushad An instruction is used. pusha targets the 16 bit register of low order of each of eight registers 32 bits for pushad. It is omissible to repeat a push instruction 8 times by pusha and pushad.
[0020] The same is said of pop, popa, and a popad instruction.
[0021] In order to fetch the disadvantageous point of repeating a push instruction that a program code becomes long and the whole instruction and to perform it, it is that activation is slow. This point is greatly improved by said pusha and pushad, when writing all of said eight registers in a stack. However, when writing in less than eight register like the case of four, or the case of six,

this point does not improve.
[0022] That is, also when all registers are operated with pusha, pushad, popa, and a popad instruction like 80386 and there is no need of evacuating and restoring all the registers, the late long instruction of this cycle must be used. In such a case, although an instruction can be managed with one instruction, the problem of becoming long produces the execution cycle of this instruction.
[0023] Moreover, in case it branches to a subroutine by a call instruction, a return instruction, etc. or returns to the called routine, shunting of a program counter and the restorative processing which are needed as the return point address are needed. In CPU of the conventional RISC method, these processings were realized by software. That is, shunting of said program counter and restorative processing were performed by executing assembler instruction (object code) which described this processing. For this reason, the call instruction and the return instruction caused the increment in object code, and in order to fetch the whole instruction and to perform, they had caused slowdown of execution speed.
[0024] The purpose of this invention is offering the information processing circuit which has the architecture which can describe efficiently the processing which deals with a stack pointer by the short instruction length, and can perform it, a microcomputer, and electronic equipment.
[0025] Again Other purposes of this invention are describing processing of register evacuation or register restoration efficiently, and offering interruption processing and the information processing circuit where the processing speed of a subroutine call return is quick, a microcomputer, and electronic equipment.
[0026]
[Means for Solving the Problem] This invention is characterized by to have the object code which makes an implied operand the stack-pointer dedicated register only used for stack pointers, and this stack-pointer dedicated register, to decode the object code of the instruction group only for stack pointers the processing based on this stack-pointer dedicated register was described to be, and to include a decode means output a control signal based on this object code, and an activation means perform said instruction group only for stack pointers based on the contents of said control signal and said stack-pointer dedicated register.
[0027] Although the thing of the program code obtained as a result of translating object code into an absolute language by the compiler generally in here is said, in this invention, it is used with the large concept which contains the program code described in the absolute language not related in the paddle by the compiler.
[0028] The information processing circuit of this invention has a stack-pointer dedicated register only for stack pointers, and it is constituted so that decode of the instruction group only for stack pointers which makes this stack-pointer dedicated register applicable to actuation, and activation may be performed.
[0029] Since said instruction group only for stack pointers has the operation code of the dedication for dealing with a stack-pointer dedicated register, it does not need the information for specifying a stack pointer for the operand of object code. In other words, said instruction group only for stack pointers makes the stack-pointer dedicated register the implied operand. For this reason, compared with the case where a stack pointer is operated using the instruction which makes one of the general-purpose registers addressing to a rate, and makes a general-purpose register applicable to actuation at a stack pointer, the instruction which deals with a stack pointer by the short instruction length can be described.
[0030] Therefore, according to this invention, the information processing circuit which can

describe and perform processing which deals with a stack pointer by the short instruction length can be offered. Moreover, the good information processing circuit of the utilization ratio of the memory which memorizes an instruction can be offered.

[0031] This invention includes the load instruction to which said instruction group only for stack pointers has transfer register specific information in object code. When said decode means decodes said load instruction and said activation means executes said load instruction, Either [at least] the data transfer from the first area to the first given given register on memory, or the data transfer from said first given register to said first given area It is characterized by carrying out based on the register address specified by the memory address specified with said stack-pointer dedicated register, and said transfer register specific information.

[0032] Said load instruction included in said instruction group only for stack pointers in here is an instruction which performs a data transfer between memory and a register, and is a concept including either [at least] the transfer to a register from memory, or the transfer to memory from a register. In addition, it is the concept which also contains address data regardless of the contents of data. A memory address means the address for pinpointing the area on the memory in the case of a transfer.

[0033] Since said load instruction has the operation code of the dedication for making a stack-pointer dedicated register applicable to actuation, it does not need the information for specifying a stack pointer for the operand of object code. For this reason, when making data transfer processing perform between the area on the memory which has the memory address related with the stack pointer, and a register, a short instruction length can describe.

[0034] This invention includes the offset information said whose load instruction is the information about the offset for specifying the address of said first area on said memory in object code, and said activation means is characterized by specifying said memory address using the contents and said offset information on said stack-pointer dedicated register.

[0035] In here, what specified the offset value by the direct immediate is sufficient as offset information, and the case where it specifies indirectly like [in the case of specifying the addresses, such as a register with which the offset value was stored] is sufficient as it. When said load instruction including offset information is executed, said memory address which is needed in case it is a transfer is specified based on the contents and offset information on a stack-pointer dedicated register.

[0036] Therefore, when making data transfer processing perform between the area on the memory which has the memory address specified based on a stack pointer and said offset information, and a register, a short instruction length can describe.

[0037] Moreover, according to this invention, also in the information processing circuit of the structure where a stack pointer always puts the word boundary, for example, it becomes possible by specifying suitable offset information to specify the area of the arbitration on a stack. For this reason, according to the size of data, it can store in a stack efficiently, and improvement in the utilization ratio of a stack can be aimed at.

[0038] Including the immediate offset information that said offset information was given by the immediate, and the data size information about the size of the given data on memory, based on said immediate offset information and said data size information, said activation means carries out the left logic shift of said immediate offset information, and generates an offset value, and this invention is characterized by to specify said memory address with the value adding the contents and said offset value of said stack-pointer dedicated register.

[0039] In here, immediate offset information means what specified the offset value by the direct

immediate. Moreover, data size information means the size of the memory top data which should be transmitted. Usually, data size is expressed with 2n (n is three or more), such as 8-bit cutting tool data, and 16 bits halfword data, 32-bit WORD data. The address on memory is given per cutting tool, halfword data are set on a halfword boundary, and WORD data are set to the word boundary. Therefore, 1 bit of the low order of the memory address of halfword data is set to 0, and 2 bits of the low order of the memory address of WORD data are set to 00. Since the address of a stack pointer has pointed out the word boundary, 1 bit of the low order of the offset value in the case of generating the memory address of halfword data is set to 0, and 2 bits of the low order of the offset value in the case of generating the memory address of WORD data are set to 00.

[0040] Moreover, a left logic shift shifts the bit string of data on the left, and means the shift in which 0 goes into the vacancy bit (shift in bit) which comes out to the right-hand side of data by shift.

[0041] According to this invention, since the left logic shift of said offset immediate information is performed based on data size, with data size, the bit of the low order decided uniquely can be omitted and immediate offset information can be described. Therefore, immediate offset information can be specified efficiently, and when it is data sizes other than a cutting tool, compared with the case where it specifies as it is, assignment becomes possible at a larger offset value.

[0042] Moreover, by using this instruction, the suitable boundary location according to the data size will be chosen at the time of the writing to the memory of data, and read-out.

[0043] This invention is characterized by changing the contents of said stack-pointer dedicated register based on said migration information, in case said decode means decodes said stack-pointer migration instruction and said activation means executes said stack-pointer migration instruction including a stack-pointer migration instruction for said instruction group only for stack pointers to have migration information in object code, and move a stack pointer to it.

[0044] Since said stack-pointer migration instruction has the operation code of the dedication for making a stack-pointer dedicated register applicable to actuation, it does not need the information for specifying a stack pointer for the operand of object code. For this reason, a short instruction length can describe to move a stack. Therefore, the amount of description of the instruction at the time of processing the data memorized by relating with the processing of data and the stack pointer which are stored in the stack is reducible.

[0045] Since migration of a stack can carry out easily according to this invention, it is effective when processing by securing stack area which is different in an especially different routine, respectively. That is, by performing processing to which a stack pointer is moved for every routine, it becomes addressable over a wide range field.

[0046] This invention is characterized by processing at least one side of the processing to which said instruction-execution means subtracts said immediate migration information from the contents of the processing adding said immediate migration information and contents of said stack-pointer dedicated register, and said stack-pointer dedicated register including the immediate migration information that said migration information was given by the immediate.

[0047] According to this invention, only a part to have specified the value of a stack pointer for immediate migration information can describe the upper part or the processing to which it is made to move caudad by the short instruction length.

[0048] This invention contains two or more registers which were able to be continuously set in order. Said instruction group only for stack pointers Either [at least] the continuation push

instruction which has two or more register specific information in object code, or a continuation pop instruction is included. Said decode means decodes either [at least] said continuation push instruction or a continuation pop instruction. When said instruction-execution means executes either [at least] said continuation push instruction or said continuation pop instruction, At least one side of the processing which carries out multiple-times pop succeeding said two or more registers from the processing which carries out a multiple-times push from said two or more registers continuously to the stack prepared in memory, and said stack It is characterized by attaining to the memory address specified according to the contents of said stack-pointer dedicated register, and carrying out based on said two or more register specific information.
[0049] As for a push, taking out data from said stack as it is pop means accumulating and storing data in the stack prepared in memory. With the processing pushed in here, and the processing which carries out pop, said storing and processing of ejection, and an update process of the stack pointer accompanying it are included. The usual information processing circuit has the push instruction for storing data and the address in a stack from the register of 1, and the pop instruction which takes out the contents of the stack to a register. This push instruction and a pop instruction update the stack pointer accompanying an exchange of the data of a register and a stack, and this exchange.
[0050] Therefore, in exchanging data etc. by two or more registers and stacks, it is necessary to carry out multiple-times activation of these instructions.
[0051] However, according to this invention, if said continuation push instruction and said continuation pop instruction are executed, the same effectiveness as the case where multiple-times activation of the case where multiple-times activation of the push instruction is carried out continuously, or the pop instruction is carried out continuously will be acquired. That is, renewal of the stack pointer accompanying an exchange and this exchange of the data between two or more registers and a stack can be performed with one instruction. For this reason, when performing the data transfer between two or more registers and a stack, it can prevent object code size increasing by repeating a push instruction or a pop instruction. Moreover, improvement in the processing speed of interruption processing and a subroutine call return can be aimed at, without avoiding that a program execution step becomes long and consuming a useless cycle.
[0052] This invention contains n general-purpose registers specified by the register numbers from 0 to n-1. One [at least] object code of said continuation push instruction and a continuation pop instruction as said two or more register specific information The last register number as which either of said register numbers was specified is included. At least one side of the processing which carries out multiple-times pop succeeding said two or more registers from the processing in which said activation means carries out a multiple-times push continuously to the stack prepared in memory from two or more registers to the register specified by said last register number from a register 0, and said stack It is characterized by carrying out based on the memory address specified according to the contents of said stack-pointer dedicated register.
[0053] Usually, when there are two or more general-purpose registers, it has the address for specifying a register. By this invention, said register is specified by the register number followed from 0 to n-1.
[0054] According to this invention, either [at least] processing which pushes data between two or more registers and memory which were followed from a register 0 to said last register number, or processing which carries out pop is performed by specifying the register number of arbitration as said last register number. Therefore, in the program execution which has structure which uses a register in an order from the register of a register number 0, shunting and restoration of a

register can be performed efficiently.
[0055] The write-in means which writes the contents of the register of two or more of said registers either given [this invention] in said activation means in the stack prepared in memory based on the memory address specified with said stack-pointer dedicated register, A count count means of writing to count the count of writing to said stack by said write-in means, A comparison means to compare the value of said count of writing counted with said count means and said two or more register specific information is included. A write-in memory address generation means by which said write-in means generates the write-in memory address for adding the first input and second input with an adder, and specifying a writing place, It controls so that the first input of said adder serves as the contents of the stack-pointer dedicated register at the time of activation initiation of the instruction only for continuation. The first [which was generated by the write-in address-generation means after it] input-control means controlled to write in and to become the address, The second input-control means which outputs the offset value when writing 1 word in the second input of said adder from said stack, The contents of the register specified with the value which subtracted said count of writing from said two or more register specific information Based on the comparison result of said comparison means, it is characterized by controlling two or more writing to said stack from a register and write-in termination including the means written in said stack based on said write-in memory address.
[0056] Moreover, this invention reads the contents of the stack by which said instruction-execution means was formed in memory based on the memory address specified with said stack-pointer dedicated register. A read-out means to store in the either given register of two or more of said registers, A count count means of read-out to count the count of read-out from said stack by said read-out means, A comparison means to compare the value of said count of read-out counted with said count means and said two or more register specific information is included. A write-in memory address generation means by which said read-out means generates the write-in memory address for adding the first input and second input with an adder, and specifying a writing place, It controls so that the first input of said adder serves as the contents of the stack-pointer dedicated register at the time of activation initiation of the instruction only for continuation. The first [which was generated by the read-out address-generation means after it] input-control means controlled to read and to become the address, The second input-control means which outputs the offset value when writing 1 word in the second input of said adder from said stack, The contents of said stack are read based on said read-out memory address. Based on the comparison result of said comparison means, it is characterized by controlling read-out and read-out termination of the contents of said stack including a read-out means to store in the register specified based on said count of writing.
[0057] If it does in this way, the restoration to two or more registers specified with the value which continued from shunting or the stack to the stack of two or more registers systematically specified with the continuous value is realizable only by the count means and easy sequence control. Therefore, since it is realizable in the information processing circuit of the small gate number, it is suitable for the microcomputer of a one chip etc.
[0058] This invention contains the program counter register only for program counters. The branch instruction said whose instruction groups only for stack pointers are the instruction which branches to a subroutine, and a return instruction from said subroutine is included. When said decode means decodes said branch instruction and said instruction-execution means executes said branch instruction, At least one side of the return to shunting of the contents of said program counter register to the second given area of the stack prepared in memory, and the program

counter register of the contents of said second area It is characterized by including the means performed based on the memory address specified with said stack-pointer dedicated register, and a means to update the contents of said stack-pointer dedicated register based on said shunting and said return.

[0059] In here, interruption processing, exception handling, a debugging manipulation routine, etc. are included with a subroutine. Therefore, with the instruction which branches to a subroutine, the software interrupt instruction for branching to an instruction, interruption processing, exception handling, a debugging manipulation routine, etc. which call a subroutine etc., a software debugging interruption instruction, etc. are included. Moreover, the return instruction from interruption processing, exception handling, a debugging manipulation routine, etc. is included in a return instruction from a subroutine.

[0060] Usually, when returning from the case where it branches to a subroutine, or said subroutine, shunting and a return of a program counter are needed.

[0061] In this invention, in case the instruction which carries out a return from the activation and the subroutine of an instruction which branch to said subroutine is executed, shunting and a return of said program counter are also performed to coincidence. That is, the information processing circuit of this invention has circuitry which can perform shunting and a return of a program counter with the instruction which branches to a subroutine, and any 1 instruction of the return instruction from said subroutine. For this reason, shunting of said program counter which is needed in connection with the return from branching and the subroutine to a subroutine, and an instruction of a return become unnecessary, and the number of instructions can be reduced. Moreover, improvement in the processing speed at the time of branching can be aimed at to other routines, such as a subroutine call return, without consuming a useless cycle.

[0062] Moreover, when a software interrupt instruction is generated, shunting and the return holding the current condition of information processing circuits, such as CPU, of a processor status register are also needed. Therefore, in a software interrupt instruction etc., it is desirable that it is made to perform shunting and a return of a processor status register to coincidence at the time of activation of this instruction.

[0063] This invention is an information processing circuit containing the stack pointer assigned to two or more registers which were able to be continuously set in order, and one of general-purpose registers. The object code of one [at least] instruction of the continuation push instruction which has two or more register specific information in object code, and a continuation pop instruction is decoded. When at least one side of a means to output a control signal based on this object code, and said continuation push instruction and said continuation pop instruction is performed, At least one side of the processing which carries out multiple-times pop succeeding said two or more registers from the processing which carries out a multiple-times push from said two or more registers continuously to the stack prepared in memory, and said stack It is characterized by including the means performed based on the memory address specified according to the contents of said control signal and said stack-pointer dedicated register, and said two or more register specific information.

[0064] This invention relates to said continuation push instruction and said continuation pop instruction in the case of using a general-purpose register as a stack pointer.

[0065] According to this invention, if said continuation push instruction and said continuation pop instruction are executed, the same effectiveness as the case where multiple-times activation of the case where multiple-times activation of the push instruction is carried out continuously, or the pop instruction is carried out continuously will be acquired. That is, renewal of the stack

pointer accompanying an exchange and this exchange of the data between two or more registers and a stack can be performed with one instruction. For this reason, when performing the data transfer between two or more registers and a stack, it can prevent object code size increasing by repeating a push instruction or a pop instruction. Moreover, improvement in the processing speed of interruption processing and a subroutine call return can be aimed at, without avoiding that a program execution step becomes long and consuming a useless cycle.

[0066] It is characterized by the information processing circuit of ******** being a RISC method.

[0067] The information processing circuit of a RISC method is designed for the purpose of miniaturizing hardware and attaining improvement in the speed. For this reason, it has many general-purpose registers and reduction of the number of instructions is aimed at by extracting an instruction set to the high thing of versatility.

[0068] Therefore, in the information processing circuit of a RISC method, the stack pointer was assigned to the general-purpose register, and when a stack pointer was treated, it was processing using the instruction set treating a general-purpose register. However, by such approach, an instruction length becomes large, and the utilization ratio of memory is not good.

[0069] According to this invention, in the information processing circuit of a RISC method, instruction lengths can be reduced and the utilization ratio of memory can be gathered.

[0070] The information processing circuit of this invention decodes a fixed-length instruction, and is characterized by performing executive operation based on this instruction.

[0071] If a fixed-length instruction is used, the time amount which decoding of an instruction takes can be shortened compared with the case where a variable-length instruction is used, and the circuit scale of an information processing circuit can be made small. When adopting a fixed-length instruction, in order to prevent making a redundant part to an instruction and to use memory efficiently, the number of bits required for each instruction has little dispersion, and its shorter possible one is desirable.

[0072] According to this invention, generally, an instruction length can shorten the instruction length of an instruction which deals with the stack pointer which tends to become long. Therefore, even if it is the case where a fixed-length instruction is adopted, it can prevent making a redundant part to an instruction, and memory can be used efficiently.

[0073] The microcomputer of this invention is characterized by including a means to perform I/O with the information processing circuit of this invention and storage means which were mentioned above, and the exterior.

[0074] According to this invention, processing speed can offer a quick microcomputer with the sufficient utilization ratio of memory.

[0075] The microcomputer of this invention is characterized by performing the program of the language which has the structure where relate with said stack pointer and the storage region of an auto variable is secured.

[0076] There is C as language which has the structure where relate with a stack pointer and the storage region of an auto variable is secured and to carry out. When the microcomputer of this invention processes the program of such language, processing speed and the utilization ratio of memory can be raised effectively.

[0077] The electronic equipment of this invention is characterized by including the microcomputer of this invention mentioned above.

[0078] According to this invention, since processing speed builds in the quick good information processing circuit of the utilization ratio of memory, cheap and highly efficient electronic

equipment can be offered.
[0079]
[Embodiment of the Invention] Hereafter, this example is explained based on a drawing.
[0080] (Example 1)
(1) By the architecture of a load store mold, CPU of configuration this example of CPU of this example executes almost all instructions in 1 cycle with a pipeline. All instructions are described by the 16-bit fixed length, and the instruction which CPU of this example processes has realized very small object code size.
[0081] Especially CPU of this example is constituted so that the instruction set of the instruction group only for stack pointers which has a register only for stack pointers in order to describe efficiently the processing which deals with a stack pointer and to perform it, and has the object code which makes this stack-pointer dedicated register an implied operand can be decoded and performed.
[0082] Drawing 1 is drawing for explaining the outline of the circuitry of CPU of this example.
[0083] A book CPU 10 contains a register set including SP14 who are a general-purpose register 11, PC12 with which the program counter is stored, the processor status register (PSR) 13, and a register only for stack pointers, an instruction decoder 20 and the immediate generation machine 22, the address adder 30, ALU40, the PC incrementer 44 and the various internal buses 72, 74, 76, and 78, the various internal signal lines 82, 84, and 86, and 88 grades.
[0084] Said instruction decoder 20 decodes the inputted object code, performs processing required in order to execute this instruction, and outputs a required control signal. In addition, this instruction decoder 20 functions also as said decode means to decode the object code of the instruction only for said stack pointers, and to output a control signal based on this instruction.
[0085] The immediate generation machine 22 generates the immediate data of 32 bits used at the time of activation based on the immediate contained in object code, or generates the constant data of 0 [required for activation of each instruction], **1, **2, and 4 [**]. The PC incrementer 44 updates the program counter stored in PC12 based on the execution cycle of an instruction. In case the address adder 30 adds the immediate data generated with the information stored in various registers, or the immediate generation vessel 22 and reads data from memory, it generates required address data. ALU40 performs math processing and logical operation.
[0086] Moreover, this CPU contains various buses and a signal line inside. PA_BUS72 and PB_BUS74 have the function to transmit the input signal of ALU40 etc. WW_BUS76 has the function which takes out the result of an operation of ALU40, and is transmitted to a general-purpose register. XA_BUS78 has the function to transmit the address data taken out from the general-purpose register 11 or SP14 etc. The IA signal line 82 transmits address data to external I_ADDR_BUS92 from each part inside CPU. The DA signal line 84 transmits address data to external D_ADDR_BUS96 from each part inside CPU. The DIN signal line 86 transmits data to each part inside CPU from D_DATA_BUS98 of the CPU exterior. The DOUT signal line 88 transmits data to external D_DATA_BUS98 from each part inside CPU. IA input change-over 83 switches the various signals (PA_BUS72, WW_BUS76, PC12, PC+2) outputted to the IA signal line 82. The DOUT input change-over 89 switches the various signals (PA_BUS72, WW_BUS76, PC12, PC+2) outputted to the DOUT signal line 88.
[0087] Moreover, said each part of CPU10 functions based on the control signal which said instruction decoder 20 outputs also as said activation means to execute an instruction and to perform the instruction group only for stack pointers based on the contents of said control signal and said stack-pointer dedicated register.

[0088] A book CPU 10 performs an exchange of the exterior and a signal through the instruction address bus (I_ADDR_BUS) 92 for the 16-bit instruction data bus (I_DATA_BUS) 94 and an instruction data access, the 32-bit data bus (D_DATA_BUS) 98, the data address bus (D_ADDR_BUS) 96 for a data access, and the control bus that is not illustrated for a control signal.
[0089] (2) the explanation of a register set in which this example carries out CPU ** -- explain a part required about the outline of the register set which CPU of this example has next.
[0090] The register set which CPU of this example has in drawing 2 is shown. CPU of this example has the register set which contains 16, and PC12, PSR13, SP14, ALR (arithmetic operation low register)15 that is not illustrated and AHR (arithmetic operation high register)16 which is not illustrated for a general-purpose register 11.
[0091] said general-purpose register 11 -- functional -- a 32-bit equivalent register -- it is -- R0 to R15, and naming ******. This general-purpose register 11 is used at the time of a data operation and address computation.
[0092] Moreover, PC12 is the incremental counter of 32 bit length, and holds the program counter which is the address of the instruction under current activation. In the text, when pointing out a register name, it is called PC, and it is called a program counter when putting the value stored in PC.
[0093] Direct access of this PC12 cannot be carried out by a load instruction etc. If a call instruction, an int instruction, interruption, and an exception occur, a program counter will be read from PC12 and will be evacuated to a stack. Thus, it will fly, if branch instruction is executed, and the point address is set as PC. It is also the same as when branching with a conditional-branching instruction. And the return point instruction address is read by a ret instruction and reti instruction from a stack, and returns to PC12 with them.
[0094] PSR (processor status register)13 is a 32-bit register to which the flag is assigned, and holds the current condition of CPU. If an int instruction, interruption, an exception, etc. occur, in case it will branch to each manipulation routine, the condition of PSR at that time is evacuated to a stack. Conversely, by activation of a reti instruction, the evacuated value returns to PSR.
[0095] SP14 is a register only for [of 32 bits] stack pointers, and the stack pointer which puts the head address of a stack is stored. In the text, when pointing out a register name, it is called SP, and it is called a stack pointer when putting the value stored in SP. However, since the stack pointer has always pointed out the boundary of WORD, 2 bits of the low order of said stack pointer are always 0.
[0096] This stack pointer is updated with activation of the various instructions included in the instruction group only for stack pointers currently prepared by this example, and generating of a trap. There are an instruction which branches to other routines, such as a call instruction and a ret instruction, as an instruction only for stack pointers which updates a stack pointer, a stack-pointer migration instruction, a pushn instruction, a popn instruction, etc. For example, if a call instruction is executed, the decrement (-4) of the stack pointer will be first carried out only for wordsize (4), and PC12 will be evacuated to a stack. Moreover, if a ret instruction is executed, the return point address will be conversely loaded to PC from a stack, and the wordsize part increment (+4) of the stack pointer will be carried out. When an int instruction is executed, or it interrupts and an exception etc. occurs, the value of PC or PSR is evacuated to a stack in the following procedures.
[0097] 1) Evacuate PC to the head address of the stack to which it is pointed out with SP=SP-42 stack pointer.

[0098] 3) Evacuate PSR to the head address of the stack to which it is pointed out with SP=SP-44 stack pointer.
[0099] If a reti instruction is executed, processing contrary to the above will be performed and CPU will return to a former condition. Thus, activation of a call instruction, a ret instruction, and an int instruction updates a stack pointer based on this activation. About the detail of each instruction, it mentions later.
[0100] Moreover, in this example, a trap means what doubled the exception generated by activation of interruption and the instruction which are generated in activation of an instruction asynchronous. If a trap occurs, CPU will read a vector table from a trap table, after evacuating a program counter (PC) and a processor status register (PSR) to a stack, and will branch to the manipulation routine corresponding to the trap. With generating of a trap, IE bit in PSR (enable [interruption]) is cleared, and generating of mask possible interruption beyond it is forbidden. In order to enable again external interruption in which a mask is possible, 1 is written in and set as IE bit of PSR using the load instruction to PSR.
[0101] A reti instruction is used in order to return to the original routine from trap processing. If a reti instruction is executed, CPU will branch to the return address while it is read from a stack in order of PSR and PC and returns PSR to the original value. In addition, there are a debugging exception, an address irregular train exception, an overflow exception, a division-by-zero exception, etc. as exception.
[0102] Explanation is omitted about ALR (arithmetic operation low register)15 and AHR (arithmetic operation high register)16.
[0103] The special registers PSR13, SP14, ALR15, and AHR16 which CPU has can perform data transfer between general-purpose registers using a load instruction. Each register has a special register number and is accessed using this number.
[0104]
A special register name Special register number The description approach processor status register of an assembler 0 % PSRSP 1 % SP arithmetic operation low register 2 % An ALR arithmetic operation high register From the lower part of the field which followed memory in the already kicked temporary storage, data accumulate a 3 %AHR (3) stack and the explanation stack about a stack pointer on ledged, and they are memorized. The stack pointer shows the address of the data memorized, the top, i.e., last, of a stack memory.
[0105] General actuation of a stack pointer is explained using drawing 3 .
[0106] 100 of drawing 3 expresses the stack area established in memory. Supposing a shadow area 102 is data stored at the end, the stack pointer stored in SP14 shows the memory address 1000 of these data. In addition, the lower part field 104 of a shadow area 102 is a field where data are already stored, and the upper part field 106 of a shadow area 102 shows the field where data will be stored from now on.
[0107] Since the stack pointer has always pointed out the boundary of WORD, when writing information in a stack, it stores information in the location where only 4 moves upwards the stack pointer with which SP14 was stored, and this stack pointer points to it. Moreover, when taking out the information stored in the stack, the information on the address which current [SP / 14] shows is taken out, and only 4 moves downward the stack pointer stored in SP14. Thus, the stack pointer shows the storing address of the information always stored in the stack finally.
[0108] (4) Although a general-purpose register is used as a stack pointer in CPU of the explanation usual RISC method about the instruction group only for stack pointers, in this example, it has SP14 who is a register only for stack pointers, and is set as the actuation object of

said instruction group only for stack pointers.
[0109] Said instruction group only for stack pointers is the generic name of two or more instructions which make an implied operand SP14 who is a register only for stack pointers, and operate it by being with SP14. This instruction group only for stack pointers includes the instruction which branches to other routines, such as SP relative load instructions (ld etc.), a stack-pointer migration instruction (add, sub), instructions (call etc.) that branch to a subroutine, and return instructions (ret etc.), a continuation push instruction (pushn), and a continuation pop instruction (popn).
[0110] Since it is the instruction only for stack pointers, I hear that that it is common in the instruction of ******** has the unnecessary information for specifying a stack pointer to object code, and it is in it. Therefore, the effectiveness that a short instruction length can describe the processing using a stack pointer efficiently is also common.
[0111] Moreover, if these instructions are used, information memorized by the stack prepared in memory can be processed efficiently. Moreover, interruption processing and processing of a subroutine call return can be performed efficiently.
[0112] Here, the busy condition of the already kicked stack and the condition of a stack pointer are explained to the example of use and memory of the instruction only for said stack pointers in case a subroutine is called using drawing 4 . The main program 500 and subroutine 520 of drawing 4 are the program described by the object code which the C compiler created. 540 shows the condition of the stack on memory. 502 shows that processing which uses general-purpose registers R0-R3 is performed. 506 shows the subroutine call instruction. The back stack pointer (SP) with which the instruction before the subroutine call instruction with which a subroutine call instruction is executed by the main program 500 (i.e., before being shown in 504) was executed presupposes that the address of ** on the stack 540 on memory was pointed out. If said subroutine call instruction is executed, control of activation will be crossed to a subroutine 520. At this time, the instruction (call instruction) which branches to the subroutine which is the instruction only for said stack pointers is executed by this example. As for the value of a stack pointer (SP), activation of this instruction stores the return address to a main program in the area 544 on the stack to which the increment only of -4 was carried out (refer to **SP of drawing 4), and this stack-pointer **SP has pointed it out automatically.
[0113] And in a subroutine 520 side, processing which transmits to a stack the value stored in the general-purpose registers R0-R3 currently used for the beginning of activation by the main program is performed. 524 shows the continuation push instruction (pushn) which is an instruction only for stack pointers which performs processing which shunts to a stack the value stored in general-purpose registers R0-R3. If this instruction is executed, the value stored in general-purpose registers R0-R3 will be continuously transmitted to a stack, and it is stored in a stack 540 as shown in 550 of drawing 4 . The stack pointer (SP) has pointed out 546 as activation of this processing finishes (524**) (**SP).
[0114] Next, in a subroutine 520, the auto variable area used by the subroutine is secured. The add instruction shown in 526 is a stack-pointer migration instruction which is an instruction only for stack pointers, and secures the stack area which is made to move a stack pointer up and is used by the subroutine 520. If this instruction is executed, only X cutting tool will move a stack pointer (SP) to the upper location 548 (**SP), and the field of the auto variable which the subroutine shown in notes 2 uses will be secured.
[0115] It is shown that 528 performs processing which used an auto variable and general-purpose registers R0-R3 by the subroutine 520. At this time, the location of a stack pointer has pointed

out **SP, and loading of an auto variable is performed by being with the load instruction only for SPs which is an instruction only for stack pointers.
[0116] 529 shows said SP load instruction which transmits the auto variable S1 stored on memory to a general-purpose register R1. Said auto variable S1 is stored in the location which has Y bytes of offset value from a stack pointer (**SP). Since a stack pointer does not move in the midst of processing of 528 of a subroutine as mentioned above, the memory address of an auto variable is specified with a stack-pointer + offset value, and can perform the exchange with a general-purpose register efficiently using the load instruction only for said SPs.
[0117] Moreover, before returning from a subroutine 520 to a main program 500, the value of the general-purpose registers R0-R3 which had shunted to the stack is returned to general-purpose registers R0-R3, and it must be made for a stack pointer to have to point out the area 544 where the return address to a main routine is stored. Therefore, the stack pointer to which it was made to move with the stack-pointer migration instruction of 526 is first returned to the location of a basis. The sub instruction shown in 530 is a stack-pointer migration instruction which is an instruction only for stack pointers, and moves a stack pointer caudad. Activation of this instruction moves a stack pointer to 546 (refer to **SP). Next, processing which restores the information 550 stored in the stack to general-purpose registers R0-R3 is performed. 532 shows the continuation pop instruction (popn) which is an instruction only for stack pointers which performs processing which transmits the information on a stack to general-purpose registers R0-R3. If this instruction is executed, the value 550 stored in the stack will be continuously transmitted to general-purpose registers R0-R3, and it is stored in a stack 540 as shown in 550 of drawing 4 . activation of this processing -- finishing (532**) -- the stack pointer has pointed out 544 (refer to **SP).
[0118] 534 shows the return instruction. If said return instruction is executed, control of activation will be crossed to a main program 500. At this time, the return instruction (ret instruction) which is an instruction only for said stack pointers is executed by this example. If this instruction is executed, it will branch to the instruction which the return address to the main program stored in the stack area which **SP points out shows, namely, will return to the instruction 507 next to a main program 500. And the value of a stack pointer moves to the head area of a stack where the increment only of +4 is carried out, and a main program 500 uses it automatically (refer to **SP of drawing 4).
[0119] The circuitry for executing explanation and this instruction of an instruction about each [said] instruction only for stack pointers below, respectively, the motion at the time of activation, etc. are explained to a detail.
[0120] (5) Create the object code a C compiler relates the field of an auto variable with a stack pointer, and it is remembered that carried out the stack-pointer relative load instruction above-mentioned. Specifically, the C compiler is the specification which secures Y bytes of field of an auto variable from a stack pointer in the place whose offset is X.
[0121] Drawing 5 is drawing for explaining the processing which transmits the auto variable on memory to a register. The auto variable a is [halfword data and auto variable d-g of WORD data and the auto variables b and c] cutting tool data. The stack pointer stored in SP shows 1000 which is the address of the head field of the field 600 where the auto variable on a stack is stored. The area on the stack which makes said stack pointer a memory address as area for storing the auto variable a The area on the stack which makes a memory address a stack pointer +2 and a stack pointer +4 as area for storing the auto variables b and c, respectively The area on the stack which makes a memory address a stack pointer +5, a stack pointer +6, a stack pointer +7, and a

stack pointer +8 as area for storing auto variable d-g, respectively is secured. And in case given processing is performed, data transfer processing is needed between the auto variable specified by the memory address of a stack-pointer + offset value, and a general-purpose register.
[0122] In CPU of this example, in order for short object code to describe such transfer processing and to perform it efficiently, the instruction set shown below is prepared as an SP relative load instruction which is one of the instructions only for stack pointers.
[0123]
ld.b %Rd [%sp+imm6] -- (1)
ld.ub %Rd [%sp+imm6] -- (2)
ld.h %Rd [%sp+imm7] -- (3)
ld.uh %Rd [%sp+imm7] -- (4)
ld.w %Rd [%sp+imm8] -- (5)
ld.b [%sp+imm6] %Rs -- (6)
ld.h [%sp+imm7] %Rs -- (7)
ld.w [%sp+imm8] %Rs -- (8)
(1) - (8) describes instruction code by the assembler. (1) is an instruction which carries out the sign escape of the cutting tool data from a stack, and is transmitted to a register. (2) is an instruction which carries out the zero escape of the cutting tool data from a stack, and is transmitted to a register. (3) is an instruction which carries out the sign escape of the halfword data from a stack, and is transmitted to a register. (4) is an instruction which carries out the zero escape of the halfword data from a stack, and is transmitted to a register. (5) is an instruction which transmits WORD data to a register from a stack. (6) is an instruction which transmits cutting tool data to a stack from a register, (7) is an instruction which transmits halfword data to a stack from a register, and (8) is an instruction which transmits WORD data to a stack from a register.
[0124] [%sp+imm6], [%sp+imm7], and [%sp+imm8] express immediate offset information, respectively, the offset value generated based on this immediate offset information at the time of activation and the value of the stack pointer stored in SP14 are added, and a memory address is generated. When the data size of the data on memory is a cutting tool, [%sp+imm7 of [%sp+imm6]] is immediate offset information when the data size of the data on memory is a halfword, in case the data size of the data on memory of [%sp+imm8] is WORD. Each of these is described as 6-bit immediate offset information 614 on object code by the business shown in drawing 6 (A) mentioned later. However, in imm7, at the time of an instruction execution, the offset value which should carry out the 1-bit left logic shift of the immediate offset information 614, and should be added to a stack pointer is generated (namely, when data size is a halfword). Moreover, in imm8, the offset value which should carry out the 2-bit left logic shift of the immediate offset information 614, and should be added to a stack pointer is generated (namely, when data size is WORD).
[0125] In here, a stack can mean the temporary storage established in memory, and the location of the auto variable (a-g) on the stack which is said memory address, for example, is shown in drawing 5 can be specified.
[0126] Drawing 6 (A) is the bit field 610 of SP relative load instruction of aforementioned (1) - (8). As shown in drawing 6 (A), SP relative load instruction has 16-bit object code including the register number 616 (4 bits) of the general-purpose register with which an actuation function serves as the operation code 612 (6 bits) which shows that it is a data transfer between memory and a register, the immediate offset information (6 bits) 614 specified by the immediate, and a

candidate for a transfer. Said operation code 612 contains a different code given according to the exception of the common code which shows that it is the load instruction which makes SP14 applicable to actuation, said data size, a sign escape, and a zero escape. Therefore, since it turns out that the stack pointer stored in SP14 by the operation code is made applicable to actuation, information about a stack pointer is not needed for the operand of object code. The immediate offset information 614 is the information for generating the offset value from the stack pointer of the data used as the candidate for a transfer. This is specified by 6 bits regardless of [having mentioned above] data size. The register number of the register with which the data with which the register number of the register with which the data with which it began to be read from a stack in aforementioned (1) - (5) are stored is written in a stack in aforementioned (6) - (8) are stored is contained in the address 616 of the register used as the candidate for a transfer.

[0127] Drawing 6 (B) shows the example of the bit field 620 of the object code of the load instruction (henceforth a general-purpose load instruction) which makes applicable to actuation the general-purpose register used when a general-purpose register is used as a stack pointer.

[0128] As shown in drawing 6 (B), the general-purpose load instruction has 20-bit object code including the operation code 622 (6 bits) an actuation function indicates it to be that it is a data transfer between memory and a general-purpose register, the immediate offset information 624 (6 bits) that it was specified by the immediate and the first register number 626 (4 bits) which specifies the general-purpose register used as a stack pointer, and the second register number 628 (4 bits) which specifies the general-purpose register with which it becomes a candidate for a transfer. In a general-purpose microcomputer, since instruction lengths are 8 bitwises, they become 24 bits or a 32-bit instruction.

[0129] Although each of drawing 6 (A) and (B) shows the object code of the instruction used when performing data transfer processing between the memory addresses and registers which are specified as a stack pointer by adding an offset value, as shown in this drawing, it can describe SP relative load instruction by short object code compared with a general-purpose load instruction.

[0130] Hereafter, (5) orders as an instruction which transmits data to a register from a stack (since it is the instruction which reads WORD data from a stack to a register). (8) orders below as an instruction which is called SP relative load instruction of WORD data read-out and which transmits data to a stack from a register (since it is the instruction which incorporates WORD data from a register to a stack). SP relative load instruction of following WORD data writing -- saying -- the configuration for taking for an example and executing these instructions and the actuation at the time of activation are explained.

[0131] A hardware configuration required in order to execute these instructions using drawing 1 first is explained. These instructions are transmitted through I_DATA_BUS94 from the external memory (ROM) 52, and are inputted into the instruction decoder 20 of CPU10. By this instruction decoder 20, an instruction is decoded and the various signals required for activation of an instruction which are not illustrated are outputted. Moreover, said immediate generation machine 22 performs the left logic shift of said immediate offset information 614 according to data size, generates the offset value used for a sign escape and zero escape deed activation if needed, and outputs it to PB_BUS74. SP14 stores the stack pointer and can output this value to XA_BUS78 connected to the input of the address adder 30. Another input of the address adder 30 is connected to PB_BUS74 which is the output of the immediate generation machine 22. The output (ADDR) of the address adder 30 is connected to external D_ADDR_BUS96 through the DA signal line 84.

[0132] A bus control unit (BCU) 60 controls I/O of data with the memory (RAM, ROM) 50 and 52 including stack area based on the various request signals (signal outputted to the external bus) outputted from CPU, and outputs READ and a WRITE control signal.
[0133] The actuation at the time of activation of SP relative load instruction of WORD data read-out is explained first.
[0134] If SP relative load instruction of WORD data read-out is executed, the value of the stack pointer stored in SP14 and the offset value which the immediate generation machine 22 generated based on said immediate offset information 614 will be added, and the memory address for memory read-out will be generated. And the information on memory is read based on this memory address, and it is transmitted to the general-purpose register specified by said register number 616 of object code.
[0135] Drawing 7 is a flow chart Fig. for explaining actuation of SP relative load instruction of WORD data read-out.
[0136] The stack pointer stored in SP14 at the beginning of activation of said instruction is outputted to XA_BUS78 (step 210). Moreover, the offset value imm which the immediate generation machine 22 generated from immediate offset information is outputted to PB_BUS74 (step 212). The address adder 30 adds the value on said XA_BUS78, and the value on said PB_BUS74, and outputs the read-out address of the memory which is a result (ADDR) to D_ADDR_BUS96 through the DA signal line 84 (step 214, step 216). And the data read-out request signal from CPU to BCU60 becomes active, and the read cycle of external memory is performed (step 218). That is, based on this request signal, data are read from memory by making said read-out address into a memory address, and said BCU60 is controlled to be outputted to D_DATA_BUS98. The data on D_DATA_BUS98 are outputted to WW_BUS76 through the DIN signal line 86 (step 220). And the value on WW_BUS76 is stored in the register (%Rd) which has the register number specified in the address (4 bits) 616 of the register (Rs/Rd) set as the transfer object of instruction code (step 222).
[0137] Next, the actuation at the time of activation of SP relative load instruction of WORD data writing is explained.
[0138] If SP relative load instruction of WORD data writing is executed, the value of the stack pointer stored in SP14 and the offset value which the immediate generation machine 22 generated based on said immediate offset information 614 will be added, and the memory address for memory writing will be generated. And the information stored in the general-purpose register specified by said register number 616 of object code is transmitted to the area on the memory specified based on this memory address.
[0139] Drawing 8 is a flow chart Fig. for explaining actuation of SP relative load instruction of WORD data writing.
[0140] The stack pointer stored in SP14 at the beginning of activation of said instruction is outputted to XA_BUS78 (step 230). Moreover, the offset value imm which the immediate generation machine 22 generated from immediate offset information is outputted to PB_BUS74 (step 232). The address adder 30 adds the value on said XA_BUS78, and the value on said PB_BUS74, and outputs the write-in address to the memory which is a result (ADDR) to D_ADDR_BUS96 through the DA signal line 84 (step 234, step 236). Moreover, the data stored in the register (%Rd) which has the register number specified in the address (4 bits) 616 of the register (Rs/Rd) set as the transfer object of instruction code are outputted to PA_BUS72 (step 238). The data on PA_BUS72 are outputted to D_DATA_BUS98 through the DOUT signal line 88 (step 240). And the data write-in request signal from CPU to BCU60 becomes active, and the

light cycle of external memory is performed (step 242). Namely, said BCU60 controls the actuation which writes the data transmitted by D_DATA_BUS98 in memory 50 by making said write-in address into a memory address based on said request signal.
[0141] (6) Stack-pointer migration instruction drawing 9 (A) - (F) is drawing for explaining the busy condition of the stack on the memory by each routine in case a program is performed over two or more routines, and the condition of a stack pointer.
[0142] The condition of the stack area on the memory at the time of activation of the given processing a of the MAIN routine 210 shown in drawing 9 (A) and the condition of a stack pointer (value stored in SP14) are shown in drawing 9 (D). 222 shows the stack area for using it by the MAIN routine 210, and the stack pointer shows the start address 232 of 222.
[0143] SUB1 routine 212 of drawing 9 (B) is a subroutine called and performed from said MAIN routine 210. The condition of the stack area on the memory at the time of activation of the given processing b213 of this SUB1 routine 212 and the condition of a stack pointer (value stored in SP) are shown in drawing 9 (E). 224 shows the stack area for using it by SUB1 routine 212, and the stack pointer shows the start address 234 of 224.
[0144] SUB2 routine 214 of drawing 9 (C) is a subroutine called and performed from said SUB1 routine 210. The condition of the stack area on the memory at the time of activation of the given processing c215 of this SUB2 routine 214 and the condition of a stack pointer (value stored in SP) are shown in drawing 9 (F). 226 shows the stack area for using it by SUB2 routine 214, and the stack pointer shows the start address 236 of 226.
[0145] Thus, when activation is performed over two or more subroutines, in order that the stack area used by each routine may move, moving to the head of a stack area which uses the value of a stack pointer by each routine in connection with it is performed.
[0146] In CPU of this example, in order for short object code to describe migration processing of such a stack pointer and to perform it efficiently, the instruction set shown below is prepared as a stack-pointer migration instruction which is one of the instructions only for stack pointers.
[0147]
add % -- sp and imm12 -- (9)
sub % -- sp and imm12 -- (10)
(9) and (10) describe instruction code by the assembler. (9) is the ADI instruction to the stack pointer stored in SP14, and (10) is the SUI instruction to the stack pointer stored in SP14. After imm12 shifts the immediate of 10 bits contained in the object code of an instruction leftward [2 bit], a zero escape is carried out, and it serves as 32 bit data, and is used for an operation with the stack pointer stored in SP14.
[0148] Drawing 10 (A) is the bit field 630 of a stack-pointer migration instruction of the above (9) and (10). As shown in drawing 10 (A), the stack-pointer migration instruction has 16-bit object code including the operation code 632 (6 bits) which shows that it is addition and subtraction of the migration information on the stack pointer with which the actuation function was stored in SP14, and the immediate migration information 634 (10 bits) specified by the immediate. Said operation code 632 contains a different code given according to the exception of the common code which shows that it is the stack-pointer migration instruction which makes SP14 applicable to actuation, addition, and subtraction. Therefore, since it turns out that the stack pointer stored in SP14 by the operation code is made applicable to actuation, information about a stack pointer is not needed for the operand of object code. The immediate migration information 634 is the information for generating the offset value which performs addition or subtraction to a stack pointer.

[0149] Drawing 10 (B) shows the example of the bit field 640 of the object code of the addition used when a general-purpose register is used as a stack pointer, and a subtraction instruction (henceforth general-purpose operation instruction).
[0150] As shown in drawing 10 (B), general-purpose operation instruction has 20-bit object code including the register number 646 (4 bits) which specifies the general-purpose register with which an actuation function serves as the operation code 642 (6 bits) which shows that it is addition and subtraction of the immediate operation information on the value of a general-purpose register, the immediate operation information 644 (10 bits) specified by the immediate, and a candidate for actuation. In a general-purpose microcomputer, since instruction lengths are 8 bitwises, they become 24 bits or a 32-bit instruction.
[0151] Although each of drawing 10 (A) and (B) is the object codes of the instruction used when performing processing which adds and subtracts an immediate to a stack pointer, as shown in this drawing, it can describe a stack-pointer migration instruction by short object code compared with general-purpose operation instruction.
[0152] Hereafter, the configuration for executing the ADI instruction (henceforth an addition stack-pointer migration instruction) to SP of (9) and the SUI instruction (henceforth a subtraction stack-pointer migration instruction) to SP of (10) and the actuation at the time of activation are explained.
[0153] A hardware configuration required in order to execute these instructions using drawing 1 first is explained. These instructions are transmitted through I_DATA_BUS94 from the external memory (ROM) 52, and are inputted into the instruction decoder 20 of CPU10. By this instruction decoder 20, an instruction is decoded and the various signals required for activation of an instruction which are not illustrated are outputted. Moreover, said immediate generation machine 22 carries out the 2-bit left logic shift of the 10 bits of said immediate migration information 634, carries out a zero escape and outputs them to PA_BUS72. SP14 stores the stack pointer and this value is outputted to XA_BUS78. XA_BUS78 is connected to PB_BUS74 used as the input of ALU40. Another input of ALU40 is connected to PA_BUS72 which is the output of the immediate generation machine 22. The output of ALU40 is connected to WW_BUS76. WW_BUS76 is connected to SP's input.
[0154] The actuation at the time of activation of an addition stack-pointer migration instruction is explained first.
[0155] When an addition stack-pointer migration instruction is executed, the value of the stack pointer stored in SP14 and the migration immediate which the immediate generation machine 22 generated based on said immediate migration information 634 are added, a new stack pointer is generated, and the value is stored in SP14.
[0156] Drawing 11 is a flow chart Fig. for explaining actuation of an addition stack-pointer migration instruction.
[0157] The stack pointer stored in SP14 at the beginning of activation of said instruction is outputted to XA_BUS78 (step 250). And the data on said XA_BUS78 are outputted to PB_BUS74 (step 252). Moreover, the migration immediate imm which the immediate generation machine 22 generated based on immediate migration information is outputted to PA_BUS72 (step 254). ALU40 adds the value on said PB_BUS74, and the value on said PA_BUS72, and outputs a result to WW_BUS76 (step 256). And the value on WW_BUS76 is inputted into SP14 (step 258).
[0158] Next, the actuation at the time of activation of a subtraction stack-pointer migration instruction is explained.

[0159] When a subtraction stack-pointer migration instruction is executed, the migration immediate which the immediate generation machine 22 generated based on said immediate migration information 634 is subtracted from the value of the stack pointer stored in SP14, a new stack pointer is generated, and the value is stored in SP14.
[0160] Drawing 12 is a flow chart Fig. for explaining actuation of a subtraction stack-pointer migration instruction.
[0161] The stack pointer stored in SP14 at the beginning of activation of said instruction is outputted to XA_BUS78 (step 260). And the data on said XA_BUS78 are outputted to PB_BUS74 (step 262). Moreover, the migration immediate imm which the immediate generation machine 22 generated based on immediate migration information is outputted to PA_BUS72 (step 264). ALU40 subtracts the value on said PA_BUS72 from the value on said PB_BUS74, and outputs a result to WW_BUS76 (step 266). And the value on WW_BUS76 is inputted into SP14 (step 268).
[0162] (7) Branch instruction drawing 13 is drawing for explaining control of the program execution by a call instruction and ret instruction. If the call instruction for branching to a subroutine SUB 310 is executed in the MAIN routine 300 as shown in drawing 13 (302), control will include a subroutine SUB 310. The ret instruction (312) is written to the last of a subroutine, and if this instruction is executed, it will return to the next instruction (304) of said call instruction (302) of the MAIN routine 300. Therefore, when shown in drawing 13 , a program will be performed in order of ******. Thus, after activation by the subroutine SUB 310 finishes, in order to perform from the next instruction (304) of return and said call instruction (302) to the MAIN routine 300, in case it branches to a subroutine SUB 310, it is necessary [it] for somewhere to memorize the address of the return point. For this reason, in the case of the branch instruction activation which branches to subroutines, such as a call instruction, as drawing 4 explained, processing which shunts the return point address to a stack is performed, and in the case of activation of the branch instruction which returns from subroutines, such as a ret instruction, processing (henceforth shunting of a program counter and restorative processing) which returns the return point address to a program counter from said stack is performed.
[0163] Since shunting of said program counter and restorative processing were realized by software, in case decision instructions, such as a call instruction, were executed, in order to perform these processings, object code (assembler instruction) was also required of CPU of the conventional RISC method. For example, in case a call instruction was executed, the object code (assembler instruction) for only wordsize (4) carrying out the decrement (-4) of the stack pointer, and storing the address of the next instruction of a call instruction in a stack based on the value of a program counter was required.
[0164] However, CPU of this example has a hardware configuration which also performs shunting of said program counter, and restorative processing together, if said call instruction and ret instruction are executed. Therefore, the object code (assembler instruction) which describes shunting of said program counter and restorative processing is not needed apart from said call instruction or ret instruction. In CPU of this example, in order to perform shunting of such said program counter, and restorative processing with one instruction, the instruction set shown below as branch instruction which is one of the instructions only for stack pointers is prepared.
[0165] call sign9 -- (11)
call %Rb -- (12)
ret -- (13)
reti -- (14)

retd -- (15)
int imm2 -- (16)
brk -- (17)
(11) - (17) describes instruction code by the assembler. (11) is PC relative subroutine call instruction, and is a call instruction which specifies the branching place address relatively and branches based on the displacement information sign9 specified with the operand by making a program counter PC into a base address. (12) is a register indirect subroutine call instruction, and is a call instruction which branches to the branching place address stored in the register specified with the operand. (13) is a return instruction from a subroutine. (14) is a return instruction from interruption or an exception-handling routine. (15) is a return instruction from a debugging manipulation routine. (16) is a software interrupt instruction. (17) is a software debugging interruption instruction.
[0166] Drawing 14 is the bit field 650 of PC relative subroutine call instruction of the above (11). It has 16-bit object code including displacement information sign9 (8 bits) 654 as which PC relative subroutine call instruction was specified by the operation code 652 (8 bits) an actuation function indicates it to be that it is the call instruction which specifies the branching place address relatively by making a program counter into a base address, and branches to a subroutine, and the immediate as shown in drawing 14 . After a logic shift is carried out to the 1-bit left at the time of activation, the sign escape of said immediate of 8 bits is carried out.
[0167] By CPU of this example, shunting to the stack of a program counter can also be performed to a call instruction execution only by the object code shown in drawing 14 .
[0168] The configuration for executing these instructions hereafter taking the case of the return instruction of (13) as PC relative subroutine call instruction of (11) and an instruction which carries out a return from a subroutine as an instruction which branches to a subroutine, and the actuation at the time of activation are explained.
[0169] A hardware configuration required in order to execute these instructions using drawing 1 first is explained. These instructions are transmitted through I_DATA_BUS94 from the external memory (ROM) 52, and are inputted into the instruction decoder 20 of CPU10. By this instruction decoder 20, an instruction is decoded and the various signals required for activation of an instruction which are not illustrated are outputted. Moreover, after said immediate generation machine 22 carries out the logic shift of said displacement information 654 to the 1-bit left, it carries out a sign escape, generates the 32-bit immediate displacement imm, and it outputs it to PB_BUS74. SP14 stores the stack pointer and can output this value to XA_BUS78 connected to the input of the address adder 30. Another input of the address adder 30 is connected to PB_BUS74 which is the output of the immediate generation machine 22. The output (ADDR) of the address adder 30 is connected to external I_ADDR_BUS92 through the IA signal line 82.
[0170] Moreover, I_ADDR_BUS92 and I_DATA_BUS94 are connected to ROM52 in which the object code of an instruction was stored, and a bus control unit (BCU) 60 outputs the READ control signal which reads the object code of said instruction from memory (ROM) 52 based on the various request signals (signal outputted to the external bus) outputted from CPU.
[0171] The actuation at the time of activation of PC relative subroutine call instruction is explained first.
[0172] If PC relative subroutine call instruction is executed, as drawing 4 explained, the value of the program counter stored in PC12 will shunt to a stack, and the decrement of the value of the stack pointer stored in SP14 will be carried out only for wordsize (4). And the branching place

address which added a program counter and said 32-bit immediate displacement to PC12, and was obtained is set.
[0173] Drawing 15 is a flow chart Fig. for explaining actuation of PC relative subroutine call instruction.
[0174] The stack pointer stored in SP14 at the beginning of activation of said instruction is outputted to XA_BUS78 (step 270). This serves as one input of an address adder, and the constant data (-4) which the immediate generation machine 22 generated serve as an input of another side of the address adder 30. and the stack which adds the value of the stack pointer on said XA_BUS78, and -4 with the address adder 30, and stores the return address -- a write-in address generation is carried out and it is outputted to WW_BUS76. Moreover, said write-in address is outputted to D_ADDR_BUS96 through the DA signal line 84. (Step 272) . Moreover, in the PC incrementer 44, the value of the program counter stored in PC12 is carried out +two, the return address is generated, and it is outputted to D_DATA_BUS98 through the DOUT signal line 88 (step 274). And the data beginning request signal from CPU to BCU60 becomes active, and the light cycle of external memory is performed (step 276). That is, said return address is stored in the stack which said BCU60 made said write-in address the memory address based on this request signal, and was prepared in memory.
[0175] And the value on WW_BUS76 is outputted to SP14 (step 278). That is, the value of a stack pointer is updated by the value carried out -four.
[0176] Next, the program counter stored in PC12 is outputted to XA_BUS78 (step 280). Moreover, after said immediate generation machine 22 carries out the logic shift of said displacement information 654 included in the object code of an instruction to the 1-bit left, it carries out a sign escape, generates the 32-bit immediate displacement imm, and it outputs it to PB_BUS74. Said address adder 30 adds the immediate displacement imm on the program counter on XA_BUS78, and PB_BUS74, generates a branch address (ADDR), and outputs it to I_ADDR_BUS92 through the IA signal line 82 (step 282). And the instruction read-out request signal from CPU to BCU60 becomes active, the read cycle of external memory (ROM) 52 is performed, and the object code of an instruction of a branching place is read (step 284).
[0177] Next, the actuation at the time of activation of a ret instruction is explained.
[0178] If a ret instruction is executed, as drawing 4 explained, the value of the program counter which had shunted to the stack will return to PC12, and the increment of the value of the stack pointer stored in SP14 will be carried out only for wordsize (4).
[0179] Drawing 16 is a flow chart Fig. for explaining actuation of a ret instruction.
[0180] Before activation of said instruction, the address of a stack with which the return point address to the called routine was stored is in the stack pointer stored in SP14 very.
[0181] The stack pointer stored in SP14 at the beginning of activation of said instruction is outputted to XA_BUS78 (step 290). And the stack pointer on XA_BUS78 is outputted to D_ADDR_BUS96 (step 292). And the data reading request signal from CPU to BCU60 becomes active, and the read cycle of external memory is performed. That is, said BCU60 reads said return address from the stack prepared in memory by making said stack pointer into a memory address based on this request signal. Said return point address read from memory (RAM) 50 is crowded for the interior of CPU through the DIN signal line 86 from D_DATA_BUS94, and is further outputted to I_ADDR_BUS92 through the IA signal line 82 from a DIN signal line (step 294). And the instruction read-out request signal from CPU to BCU60 becomes active, the read cycle of external memory (ROM) 52 is performed, and the object code of an instruction of the return point address is read (step 296).

[0182] And the value of the stack pointer on XA_BUS78 serves as one input of said address adder 30. Moreover, the constant data (+4) which the immediate generation machine 22 generated serve as an input of another side of the address adder 30. And the value of the stack pointer on said XA_BUS78 and +4 are added with the address adder 30, the address of the head area of the stack area which the routine of the return point secured is generated, and it is outputted to WW_BUS76 (step 298). And said address on WW_BUS76 (address of the head area of the stack area which the routine of the return point secured) is outputted to SP14 (step 300).
[0183] (8) Without especially CPU of a RISC method having many general-purpose registers in the interior in order [latest] to raise the engine performance, and accessing memory, as the continuation push instruction (pùshn) and the continuation pop instruction (popn) carried out the explanation above-mentioned, it is constituted so that many may be processed at a high speed inside CPU. This example also has 16 general-purpose registers in the interior, and is attaining improvement in the speed of processing. However, if it has many internal registers in this way, the number of registers which should be evacuated will increase in the case of interruption processing or processing of the register evacuation at the time of a subroutine call, and restoration.
[0184] When shunting and restoration of such a register were performed, the push instruction which stores in a stack the contents specified by the address part, and the pop instruction which takes out the contents of the stack to a register were used conventionally. The motion at the time of a common push instruction and a pop instruction is explained here.
[0185] Drawing 17 (A) and (B) are drawings having shown typically the motion at the time of a push instruction execution, and drawing 18 (A) and (B) are drawings having shown typically the motion at the time of a pop instruction execution. The motion in the case of performing a data transfer between two or more general-purpose registers and a stack is explained using drawing 17 (A), (B), and drawing 18 (A) and (B).
[0186] Drawing 17 (A) shows the motion when 'push R1' which is the instruction which writes out the contents of the general-purpose register R1 to a stack is performed. SP's14 contents are updated by the value by which 4 was subtracted from the current value at the time of activation of this instruction (1000 is updated by 996 as shown in drawing 17 (A)). And the contents a of the general-purpose register R1 are written in 996 which is the memory address which SP's14 updated stack pointer shows.
[0187] Drawing 17 (B) shows the motion when 'push R2' which is the instruction which writes out the contents of the general-purpose register R2 to a stack further is performed. SP's14 contents are updated by the value by which 4 was subtracted from the current value at the time of activation of this instruction (996 is updated by 992 as shown in drawing 17 (B)). And the contents b of the general-purpose register R2 are written in 992 which is the memory address which SP's14 updated stack pointer shows.
[0188] Drawing 18 (A) shows the motion when 'pop R2' which is the instruction which takes out the contents of the stack to a general-purpose register R2 is performed. At the time of activation of this instruction, the contents b stored in the memory address 992 which SP's14 stack pointer shows are taken out, and it is stored in a general-purpose register R2. And SP's14 contents are updated by the value with which 4 was added to the current value (drawing 18 (A) it is updated by 996 after activation that it was front [activation] 992 so that it might be shown).
[0189] Drawing 18 (B) shows the motion when 'pop R1' which is the instruction which takes out the contents of the stack to a general-purpose register R1 further is performed. At the time of

activation of this instruction, the contents a stored in the memory address 996 which SP's14 stack pointer shows are taken out, and it is stored in a general-purpose register R1. And SP's14 contents are updated by the value with which 4 was added to the current value (it is updated by 1000 after activation that it was front [activation] 996 as shown in drawing 18 (B)).
[0190] Thus, it was required to repeat a push instruction and a pop instruction two or more times, and to execute them conventionally, when performing a data transfer by two or more general-purpose registers and stacks. It is because only one register operated the push instruction and the pop instruction with one instruction.
[0191] Therefore, when restoration processing from shunting and the stack to a stack was performed to many registers, increase of the size of object code was caused by the increment in the number of instructions. Moreover, the program execution step also increased and delay of program execution time amount and processing actuation was caused.
[0192] There The instruction set shown below is prepared in CPU of this example.
[0193] pushn %Rs -- (18)
popn %Rd -- (19)
(18) shows description of the assembler of a continuation push instruction, and is an instruction which pushes continuously the contents of n general-purpose registers (n is the natural number of 1 to 16) from %Rs to R0 to a stack. (19) shows description of the assembler of a continuation pop instruction, and is an instruction which carries out pop [of the n WORD data (n is the natural number of 1 to 16)] from a stack to the general-purpose register from %Rd to R0 continuously. Each of pushn(s) and popn instructions consists of an operation code and an operand. % Rs shows the register number of Rs in the case of writing from register %Rs to R0 in a stack with the operand of a pushn instruction. % Rd shows the register number of Rd in the case of bringing data from a stack to the register from R0 to %Rd with the operand of a popn instruction.
[0194] The bit map of pushn and a popn instruction is shown in drawing 19 . The code which shows %Rs or %Rd goes into the 4-bit low-ranking field, and the register of the arbitration of the general-purpose register which has 16 can be specified. When carrying out a data transfer between a special register and a stack, it carries out through a general-purpose register.
[0195] Drawing 20 is a block diagram for explaining the hardware configuration for executing a continuation push instruction (pushn) and a continuation pop instruction (popn). A part required for explanation of a continuation push instruction (pushn) and a continuation pop instruction (popn) is taken out from drawing 1 , and it has the composition of having added the part required for explanation. The number same about what points out the same part as drawing 1 is attached. 11 is 16 general-purpose registers of R15 out of [R0] drawing. A general-purpose register 11 performs I/O of a data bus (D_DATA_BUS) 98 and data. Moreover, the register-select address signal 54 which chooses a register comes from the control circuit block 45. As for 14, the stack pointer is stored by SP. SP's14 value can be outputted to the internal address bus (XA_BUS) 78 connected with an address bus (D_ADDR_BUS) 96 at the input of the 32-bit address adder 30. Another input of the address adder 30 is connected to the offset signal 24 outputted from the control circuit block 45. The output of the address adder 30 is latched by latch (Add_LT) 32, and is further outputted to XA_BUS78 or WW_BUS76. WW_BUS76 is connected to SP's14 input. The 4-bit counter (countx) 46 is in the control circuit block 45, and the number of registers to transmit is counted. Moreover, although not shown by drawing 20 during the control circuit block 45, while there is an instruction register and holding pushn, O ** land %Rs of a popn instruction, and %Rd, the control signal according to each instruction is outputted by the

instruction decoder. 60 in drawing controls I/O of data with the memory (RAM) 50 including external stack area by the bus control unit (BCU), and outputs READ and a WRITE control signal.
[0196] The actuation at the time of activation of a continuation push instruction (pushn) is explained first.
[0197] If a continuation push instruction (pushn) is executed, as drawing 4 explained, the contents stored in the general-purpose register from the general-purpose register of the register number of %Rs to a general-purpose register R0 will be continuously pushed by the stack.
[0198] Drawing 21 is a flow chart Fig. for explaining actuation of a pushn instruction.
[0199] offset of the offset signal 24 is set to -four to the beginning of activation of a pushn instruction. A counter (countx) 46 is cleared by zero and the stack-pointer value stored in SP14 is outputted to XA_BUS78 (step 100).
[0200] Next, the address adder 30 adds the value on XA_BUS78, and -4, and puts a result into latch (Add_LT) 32 (step 101).
[0201] Latch's (Add_LT) 32 value is outputted to address bus D_ADDR_BUS96. In the control circuit block 45, the difference of %Rs and a counter (countx) 46 held at 4 bits of low order of an instruction register is calculated, and a result is outputted to the register-select address signal 54. The register chosen by 54 is put on a data bus (D_DATA_BUS) 98, and write-in actuation of external memory (RAM) 50 is carried out (step 102).
[0202] Step 103 compares %Rs with a counter (countx) 46. When equal, it has completed, and the writing to the external memory of a register writes latch's (Add_LT) 32 value in SP14 through WW_BUS76, and finishes activation of pushn (step 104).
[0203] When not equal, a counter (countx) is carried out plus 1, latch's (Add_LT) 32 value is outputted to XA_BUS78, and the processing after step 101 is repeated (step 105).
[0204] Next, the actuation at the time of activation of a continuation pop instruction (popn) is explained.
[0205] If a continuation pop instruction (popn) is executed, as drawing 4 explained, the contents of the stack will be continuously pushed by the general-purpose register of the register number of %Rd from a general-purpose register R0.
[0206] Drawing 22 is a flow chart Fig. for explaining actuation of a popn instruction.
[0207] The offset signal 24 is set to +4 to the beginning of a popn instruction. The zero clear of the counter (countx) 46 is carried out, and the value of SP's14 stack pointer is outputted to XA_BUS78 and an address bus (D_ADDR_BUS) 96 (step 110).
[0208] Next, the address adder 30 adds the value on XA_BUS78, and +4, and puts a result into latch (Add_LT) 32 (step 111).
[0209] Next, the read cycle of external memory is performed. The led data are written in a general-purpose register 11 through a data bus (D_DATA_BUS) 98. At this time, a counter (countx) 46 is outputted to the register-select address signal 54 by the control circuit block 45 (step 112).
[0210] Step 113 compares %Rs with a counter (countx) 46. When equal, it has completed, and the writing to the external memory of a register writes latch's (Add_LT) 32 value in SP14 through WW_BUS76, and finishes activation of popn (step 114).
[0211] When not equal, a counter (countx) is carried out plus 1, latch's (Add_LT) 32 value is outputted to XA_BUS78 and an address bus (D_ADDR_BUS) 96, and the processing after step 111 is repeated (step 115).
[0212] Thus, the register from %Rs to R0 can be written in a stack by 'pushn %Rs', and the data

of the required number can be returned to the register from R0 to %Rd from a stack by 'popn %Rd'. For example, from the register R3 to R0 can be pushed by one instruction by activation of 'pushn%R3'.
[0213] Still more effective effectiveness occurs by adding one constraint to how using a register in here. That is, although shunting and restoration of a register are needed especially when the programs at interruption processing, the time of a subroutine call, etc. branch to other routines, it is desirable to go a register by routines called, such as an interruption routine besides this time or a subroutine, using order from R0. If it does in this way, the register which does not have the need for evacuation in Rd or Rs is not contained from R0, but shunting or a return of a register can be efficiently performed using a pushn instruction and a popn instruction. Since it has the same function and there is no constraint in an instruction the direction using a register, such constraint does not have any problem and each register can fill it with this example.
[0214] Therefore, by using a pushn instruction and popn instruction of this example, pushn and popn are one instruction respectively and the evacuation to the stack in memory from a register and restoration of the data from a stack to a register can perform them. For this reason, object code size and a program execution step are made into the minimum, and only one data transfer of an instruction fetch and a required count is made to require, and an execution cycle can be performed in the minimum cycle. Thereby, processing of an interrupt handler or a subroutine can also attain improvement in the speed.
[0215] On the other hand, the components which realize this are only a count means and easy sequence control, and it can realize with the small gate number and they fit the microcomputer of a one chip.
[0216] Moreover, although this example explained the configuration for performing the continuation push instruction (pushn) at the time of using the register SP 14 only for stack pointers, and a continuation pop instruction (popn), also when using a general-purpose register as a stack pointer, it can apply.
[0217] (Example 2) Drawing 23 is the hardware block diagram of the microcomputer of this example.
[0218] This microcomputer 2 is a 32-bit microcontroller. CPU10, ROM52, RAM50 and the RF oscillator circuit 910, the low frequency oscillator circuit 920, a reset circuit 930, the 950 or 8 bit programmable timer 960 of 940 or 16 bit programmable timers of prescalers, the clock timer 970, Intelligent DMA980, a high speed DMA 990, the interruption controller 800, serial interface 810, a bus control unit (BCU) 60, A/D converter 830, D/A converter 840, input port 850, an output port 860, I/O Port 870 and the various buses 92, 94, 96, and 98 which connect them, and various pin 890 grades are included.
[0219] Said CPU10 performs decode of various kinds of instructions only for stack pointers which had and mentioned above SP who is a stack-pointer dedicated register, and activation. This CPU10 has the configuration of an example 1 mentioned above, and functions as said decode means and said activation means.
[0220] Therefore, by the short instruction length, the processing which deals with a stack pointer can be memorized efficiently, and the microcomputer of this example can perform it.
[0221] Moreover, processing of register evacuation or register restoration can be memorized efficiently, and interruption processing and processing of a subroutine call return can be performed at a high speed.
[0222] The microcomputer of this invention is applicable to personal computer peripheral devices, such as a printer, and various kinds of electronic equipment, such as a pocket device. If

it is made this appearance, since the utilization ratio of memory can build well the information processing circuit which can perform processing in a high speed with an easy configuration, cheap and highly efficient electronic equipment can be offered.
[0223] In addition, what [not only] was explained in the above-mentioned example but various deformation implementation is possible for this invention.

TECHNICAL FIELD

[Field of the Invention] This invention relates to the electronic equipment constituted using the microcomputer which contains an information processing circuit and said information processing circuit, and this microcomputer.

TECHNICAL PROBLEM

Background Art and Problem(s) to be Solved by the Invention] Conventionally, in the microcomputer of a RISC method which can process 32-bit data, the fixed-length instruction of 32-bit width of face was used. The reason is that the time amount which decoding of an instruction takes can be shortened compared with the case where a variable-length instruction is used, and it can make the circuit scale of a microcomputer small if a fixed-length instruction is used.
[0003] However, also in a 32-bit microcomputer, there are also many instructions which are not needed especially 32 bits. Therefore, if 32 bits describes all instructions, the instructions which a redundant part produces in an instruction will increase in number, and the utilization ratio of memory will worsen.
[0004] then, this invention -- the person was performing examination about the microcomputer which processes the fixed-length instruction of bit width of face shorter than the bit width of face of the data which can be processed in order to raise the utilization ratio of memory, without complicating a control circuit.
[0005] However, if 32 bit fixed length's instruction is only made into a 16-bit fixed length, for example, the following troubles will arise.
[0006] That is, in the microcomputer of a RISC method, in order to think the versatility of processing and an instruction set as important, when processing which deals with a stack pointer was performed, said processing was performed using the instruction which makes a general-purpose register applicable to actuation, using one of general-purpose registers as a stack pointer. Therefore, assignment of the general-purpose register currently used as a stack pointer is needed for that of said instruction at the time of describing such processing.
[0007] For example, when the instruction which makes a general-purpose register applicable to actuation describes the processing which transmits the data specified by the memory address which applied the given offset value to the stack pointer to a given register, said offset value, the information which specifies said given register, and the information which specifies the register currently used as a stack pointer are needed for the object code of the instruction.
[0008] Thus, since the information specified by object code increases when describing the processing which deals with a stack pointer with the instruction which makes a general-purpose

register applicable to actuation, it becomes difficult to describe the contents of the instruction by the 16-bit fixed length. Also especially into an instruction, if an instruction length is made into 32 bits here, since an instruction also has many instructions which are not needed 32 bits, the instructions which produce a redundant part will increase in number, and aggravation of the utilization ratio of memory will be invited to an instruction.
[0009] Moreover, since the memory which stores it is also too much needed if an instruction length becomes long, it is desirable that not only a fixed-length instruction but an instruction length can be shortened from a viewpoint of efficient use of memory.
[0010] Moreover, since the instructions which deal with a stack pointer increase in number when performing the program described in the language in which it relates with a stack pointer, for example like C, and the storage region of an auto variable is secured, it is desirable to describe efficiently the instruction which deals with a stack pointer, and to perform it.
[0011] Then, when processing which deals with a stack pointer was performed, by the shortest possible instruction length, the contents of an instruction were described and architecture which can be performed was desired.
[0012] Moreover, especially CPU of a RISC method came to have many general-purpose registers in the interior in order [latest] to raise the engine performance. It is because many processings are made to a high speed inside CPU, without accessing memory by having many registers in the interior. Thus, if it has many internal registers, the number of registers which should be evacuated will increase in the case of interruption processing or processing of the register evacuation at the time of a subroutine call, and restoration.
[0013] The conventional example is explained taking the case of shunting of the register which uses it abundantly in case going into a subroutine also in a stack system instruction **** is sufficient hereafter and it carries out, and return instruction.
[0014] Usually, the instruction set of a microcomputer shunts to the stack which prepared the register of CPU in memory, or has the instruction for restoring. There is a thing with an instruction of dedication for that or a thing with a register indirect-addressing instruction.
[0015] As a technique about the instruction for [said] shunting or restoring, "80386 Pro gramine" (John H.Crawford Patric P.Gelsigner work Iwatani ** translation) has the following description about 80386 of Intel.
[0016] That is, it is an instruction which writes a register in a stack. There are push, pusha, and pushad and there are pop, popa, and popad as an instruction for returning data to a register from a stack.
[0017] When writing a register in a stack with a push instruction push EAX A register is specified as an operand like. This is the case of the 32-bit register EAX. It is push when writing Registers EAX, ECX, EDX, and EBX to a stack altogether. EAXpush ECXpush EDXpush A push instruction is repeated like EBX.
[0018] Thus, if it is operating one register at a time with push and a pop instruction, the size of object code will become large, and a program execution step will also increase, therefore program execution time amount and processing actuation will become slower.
[0019] then, all eight general-purpose registers that 80386 has -- a stack -- writing in -- pusha or -- pushad An instruction is used. pusha targets the 16 bit register of low order of each of eight registers 32 bits for pushad. It is omissible to repeat a push instruction 8 times by pusha and pushad.
[0020] The same is said of pop, popa, and a popad instruction.
[0021] In order to fetch the disadvantageous point of repeating a push instruction that a program

code becomes long and the whole instruction and to perform it, it is that activation is slow. This point is greatly improved by said pusha and pushad, when writing all of said eight registers in a stack. However, when writing in less than eight register like the case of four, or the case of six, this point does not improve.
[0022] That is, also when all registers are operated with pusha, pushad, popa, and a popad instruction like 80386 and there is no need of evacuating and restoring all the registers, the late long instruction of this cycle must be used. In such a case, although an instruction can be managed with one instruction, the problem of becoming long produces the execution cycle of this instruction.
[0023] Moreover, in case it branches to a subroutine by a call instruction, a return instruction, etc. or returns to the called routine, shunting of a program counter and the restorative processing which are needed as the return point address are needed. In CPU of the conventional RISC method, these processings were realized by software. That is, shunting of said program counter and restorative processing were performed by executing assembler instruction (object code) which described this processing. For this reason, the call instruction and the return instruction caused the increment in object code, and in order to fetch the whole instruction and to perform, they had caused slowdown of execution speed.
[0024] The purpose of this invention is offering the information processing circuit which has the architecture which can describe efficiently the processing which deals with a stack pointer by the short instruction length, and can perform it, a microcomputer, and electronic equipment.
[0025] Again Other purposes of this invention are describing processing of register evacuation or register restoration efficiently, and offering interruption processing and the information processing circuit where the processing speed of a subroutine call return is quick, a microcomputer, and electronic equipment.

MEANS

[Means for Solving the Problem] This invention is characterized by to have the object code which makes an implied operand the stack-pointer dedicated register only used for stack pointers, and this stack-pointer dedicated register, to decode the object code of the instruction group only for stack pointers the processing based on this stack-pointer dedicated register was described to be, and to include a decode means output a control signal based on this object code, and an activation means perform said instruction group only for stack pointers based on the contents of said control signal and said stack-pointer dedicated register.
[0027] Although the thing of the program code obtained as a result of translating object code into an absolute language by the compiler generally in here is said, in this invention, it is used with the large concept which contains the program code described in the absolute language not related in the paddle by the compiler.
[0028] The information processing circuit of this invention has a stack-pointer dedicated register only for stack pointers, and it is constituted so that decode of the instruction group only for stack pointers which makes this stack-pointer dedicated register applicable to actuation, and activation may be performed.
[0029] Since said instruction group only for stack pointers has the operation code of the dedication for dealing with a stack-pointer dedicated register, it does not need the information for specifying a stack pointer for the operand of object code. In other words, said instruction group

only for stack pointers makes the stack-pointer dedicated register the implied operand. For this reason, compared with the case where a stack pointer is operated using the instruction which makes one of the general-purpose registers addressing to a rate, and makes a general-purpose register applicable to actuation at a stack pointer, the instruction which deals with a stack pointer by the short instruction length can be described.

[0030] Therefore, according to this invention, the information processing circuit which can describe and perform processing which deals with a stack pointer by the short instruction length can be offered. Moreover, the good information processing circuit of the utilization ratio of the memory which memorizes an instruction can be offered.

[0031] This invention includes the load instruction to which said instruction group only for stack pointers has transfer register specific information in object code. When said decode means decodes said load instruction and said activation means executes said load instruction, Either [at least] the data transfer from the first area to the first given given register on memory, or the data transfer from said first given register to said first given area It is characterized by carrying out based on the register address specified by the memory address specified with said stack-pointer dedicated register, and said transfer register specific information.

[0032] Said load instruction included in said instruction group only for stack pointers in here is an instruction which performs a data transfer between memory and a register, and is a concept including either [at least] the transfer to a register from memory, or the transfer to memory from a register. In addition, it is the concept which also contains address data regardless of the contents of data. A memory address means the address for pinpointing the area on the memory in the case of a transfer.

[0033] Since said load instruction has the operation code of the dedication for making a stack-pointer dedicated register applicable to actuation, it does not need the information for specifying a stack pointer for the operand of object code. For this reason, when making data transfer processing perform between the area on the memory which has the memory address related with the stack pointer, and a register, a short instruction length can describe.

[0034] This invention includes the offset information said whose load instruction is the information about the offset for specifying the address of said first area on said memory in object code, and said activation means is characterized by specifying said memory address using the contents and said offset information on said stack-pointer dedicated register.

[0035] In here, what specified the offset value by the direct immediate is sufficient as offset information, and the case where it specifies indirectly like [in the case of specifying the addresses, such as a register with which the offset value was stored] is sufficient as it. When said load instruction including offset information is executed, said memory address which is needed in case it is a transfer is specified based on the contents and offset information on a stack-pointer dedicated register.

[0036] Therefore, when making data transfer processing perform between the area on the memory which has the memory address specified based on a stack pointer and said offset information, and a register, a short instruction length can describe.

[0037] Moreover, according to this invention, also in the information processing circuit of the structure where a stack pointer always puts the word boundary, for example, it becomes possible by specifying suitable offset information to specify the area of the arbitration on a stack. For this reason, according to the size of data, it can store in a stack efficiently, and improvement in the utilization ratio of a stack can be aimed at.

[0038] Including the immediate offset information that said offset information was given by the

immediate, and the data size information about the size of the given data on memory, based on said immediate offset information and said data size information, said activation means carries out the left logic shift of said immediate offset information, and generates an offset value, and this invention is characterized by to specify said memory address with the value adding the contents and said offset value of said stack-pointer dedicated register.

[0039] In here, immediate offset information means what specified the offset value by the direct immediate. Moreover, data size information means the size of the memory top data which should be transmitted. Usually, data size is expressed with 2n (n is three or more), such as 8-bit cutting tool data, and 16 bits halfword data, 32-bit WORD data. The address on memory is given per cutting tool, halfword data are set on a halfword boundary, and WORD data are set to the word boundary. Therefore, 1 bit of the low order of the memory address of halfword data is set to 0, and 2 bits of the low order of the memory address of WORD data are set to 00. Since the address of a stack pointer has pointed out the word boundary, 1 bit of the low order of the offset value in the case of generating the memory address of halfword data is set to 0, and 2 bits of the low order of the offset value in the case of generating the memory address of WORD data are set to 00.

[0040] Moreover, a left logic shift shifts the bit string of data on the left, and means the shift in which 0 goes into the vacancy bit (shift in bit) which comes out to the right-hand side of data by shift.

[0041] According to this invention, since the left logic shift of said offset immediate information is performed based on data size, with data size, the bit of the low order decided uniquely can be omitted and immediate offset information can be described. Therefore, immediate offset information can be specified efficiently, and when it is data sizes other than a cutting tool, compared with the case where it specifies as it is, assignment becomes possible at a larger offset value.

[0042] Moreover, by using this instruction, the suitable boundary location according to the data size will be chosen at the time of the writing to the memory of data, and read-out.

[0043] This invention is characterized by changing the contents of said stack-pointer dedicated register based on said migration information, in case said decode means decodes said stack-pointer migration instruction and said activation means executes said stack-pointer migration instruction including a stack-pointer migration instruction for said instruction group only for stack pointers to have migration information in object code, and move a stack pointer to it.

[0044] Since said stack-pointer migration instruction has the operation code of the dedication for making a stack-pointer dedicated register applicable to actuation, it does not need the information for specifying a stack pointer for the operand of object code. For this reason, a short instruction length can describe to move a stack. Therefore, the amount of description of the instruction at the time of processing the data memorized by relating with the processing of data and the stack pointer which are stored in the stack is reducible.

[0045] Since migration of a stack can carry out easily according to this invention, it is effective when processing by securing stack area which is different in an especially different routine, respectively. That is, by performing processing to which a stack pointer is moved for every routine, it becomes addressable over a wide range field.

[0046] This invention is characterized by processing at least one side of the processing to which said instruction-execution means subtracts said immediate migration information from the contents of the processing adding said immediate migration information and contents of said stack-pointer dedicated register, and said stack-pointer dedicated register including the

immediate migration information that said migration information was given by the immediate.
[0047] According to this invention, only a part to have specified the value of a stack pointer for immediate migration information can describe the upper part or the processing to which it is made to move caudad by the short instruction length.
[0048] This invention contains two or more registers which were able to be continuously set in order. Said instruction group only for stack pointers Either [at least] the continuation push instruction which has two or more register specific information in object code, or a continuation pop instruction is included. Said decode means decodes either [at least] said continuation push instruction or a continuation pop instruction. When said instruction-execution means executes either [at least] said continuation push instruction or said continuation pop instruction, At least one side of the processing which carries out multiple-times pop succeeding said two or more registers from the processing which carries out a multiple-times push from said two or more registers continuously to the stack prepared in memory, and said stack It is characterized by attaining to the memory address specified according to the contents of said stack-pointer dedicated register, and carrying out based on said two or more register specific information.
[0049] As for a push, taking out data from said stack as it is pop means accumulating and storing data in the stack prepared in memory. With the processing pushed in here, and the processing which carries out pop, said storing and processing of ejection, and an update process of the stack pointer accompanying it are included. The usual information processing circuit has the push instruction for storing data and the address in a stack from the register of 1, and the pop instruction which takes out the contents of the stack to a register. This push instruction and a pop instruction update the stack pointer accompanying an exchange of the data of a register and a stack, and this exchange.
[0050] Therefore, in exchanging data etc. by two or more registers and stacks, it is necessary to carry out multiple-times activation of these instructions.
[0051] However, according to this invention, if said continuation push instruction and said continuation pop instruction are executed, the same effectiveness as the case where multiple-times activation of the case where multiple-times activation of the push instruction is carried out continuously, or the pop instruction is carried out continuously will be acquired. That is, renewal of the stack pointer accompanying an exchange and this exchange of the data between two or more registers and a stack can be performed with one instruction. For this reason, when performing the data transfer between two or more registers and a stack, it can prevent object code size increasing by repeating a push instruction or a pop instruction. Moreover, improvement in the processing speed of interruption processing and a subroutine call return can be aimed at, without avoiding that a program execution step becomes long and consuming a useless cycle.
[0052] This invention contains n general-purpose registers specified by the register numbers from 0 to n-1. One [at least] object code of said continuation push instruction and a continuation pop instruction as said two or more register specific information The last register number as which either of said register numbers was specified is included. At least one side of the processing which carries out multiple-times pop succeeding said two or more registers from the processing in which said activation means carries out a multiple-times push continuously to the stack prepared in memory from two or more registers to the register specified by said last register number from a register 0, and said stack It is characterized by carrying out based on the memory address specified according to the contents of said stack-pointer dedicated register.
[0053] Usually, when there are two or more general-purpose registers, it has the address for specifying a register. By this invention, said register is specified by the register number followed

from 0 to n-1.
[0054] According to this invention, either [at least] processing which pushes data between two or more registers and memory which were followed from a register 0 to said last register number, or processing which carries out pop is performed by specifying the register number of arbitration as said last register number. Therefore, in the program execution which has structure which uses a register in an order from the register of a register number 0, shunting and restoration of a register can be performed efficiently.
[0055] The write-in means which writes the contents of the register of two or more of said registers either given [this invention] in said activation means in the stack prepared in memory based on the memory address specified with said stack-pointer dedicated register, A count count means of writing to count the count of writing to said stack by said write-in means, A comparison means to compare the value of said count of writing counted with said count means and said two or more register specific information is included. A write-in memory address generation means by which said write-in means generates the write-in memory address for adding the first input and second input with an adder, and specifying a writing place, It controls so that the first input of said adder serves as the contents of the stack-pointer dedicated register at the time of activation initiation of the instruction only for continuation. The first [which was generated by the write-in address-generation means after it] input-control means controlled to write in and to become the address, The second input-control means which outputs the offset value when writing 1 word in the second input of said adder from said stack, The contents of the register specified with the value which subtracted said count of writing from said two or more register specific information Based on the comparison result of said comparison means, it is characterized by controlling two or more writing to said stack from a register and write-in termination including the means written in said stack based on said write-in memory address.
[0056] Moreover, this invention reads the contents of the stack by which said instruction-execution means was formed in memory based on the memory address specified with said stack-pointer dedicated register. A read-out means to store in the either given register of two or more of said registers, A count count means of read-out to count the count of read-out from said stack by said read-out means, A comparison means to compare the value of said count of read-out counted with said count means and said two or more register specific information is included. A write-in memory address generation means by which said read-out means generates the write-in memory address for adding the first input and second input with an adder, and specifying a writing place, It controls so that the first input of said adder serves as the contents of the stack-pointer dedicated register at the time of activation initiation of the instruction only for continuation. The first [which was generated by the read-out address-generation means after it] input-control means controlled to read and to become the address, The second input-control means which outputs the offset value when writing 1 word in the second input of said adder from said stack, The contents of said stack are read based on said read-out memory address. Based on the comparison result of said comparison means, it is characterized by controlling read-out and read-out termination of the contents of said stack including a read-out means to store in the register specified based on said count of writing.
[0057] If it does in this way, the restoration to two or more registers specified with the value which continued from shunting or the stack to the stack of two or more registers systematically specified with the continuous value is realizable only by the count means and easy sequence control. Therefore, since it is realizable in the information processing circuit of the small gate number, it is suitable for the microcomputer of a one chip etc.

[0058] This invention contains the program counter register only for program counters. The branch instruction said whose instruction groups only for stack pointers are the instruction which branches to a subroutine, and a return instruction from said subroutine is included. When said decode means decodes said branch instruction and said instruction-execution means executes said branch instruction, At least one side of the return to shunting of the contents of said program counter register to the second given area of the stack prepared in memory, and the program counter register of the contents of said second area It is characterized by including the means performed based on the memory address specified with said stack-pointer dedicated register, and a means to update the contents of said stack-pointer dedicated register based on said shunting and said return.
[0059] In here, interruption processing, exception handling, a debugging manipulation routine, etc. are included with a subroutine. Therefore, with the instruction which branches to a subroutine, the software interrupt instruction for branching to an instruction, interruption processing, exception handling, a debugging manipulation routine, etc. which call a subroutine etc., a software debugging interruption instruction, etc. are included. Moreover, the return instruction from interruption processing, exception handling, a debugging manipulation routine, etc. is included in a return instruction from a subroutine.
[0060] Usually, when returning from the case where it branches to a subroutine, or said subroutine, shunting and a return of a program counter are needed.
[0061] In this invention, in case the instruction which carries out a return from the activation and the subroutine of an instruction which branch to said subroutine is executed, shunting and a return of said program counter are also performed to coincidence. That is, the information processing circuit of this invention has circuitry which can perform shunting and a return of a program counter with the instruction which branches to a subroutine, and any 1 instruction of the return instruction from said subroutine. For this reason, shunting of said program counter which is needed in connection with the return from branching and the subroutine to a subroutine, and an instruction of a return become unnecessary, and the number of instructions can be reduced. Moreover, improvement in the processing speed at the time of branching can be aimed at to other routines, such as a subroutine call return, without consuming a useless cycle.
[0062] Moreover, when a software interrupt instruction is generated, shunting and the return holding the current condition of information processing circuits, such as CPU, of a processor status register are also needed. Therefore, in a software interrupt instruction etc., it is desirable that it is made to perform shunting and a return of a processor status register to coincidence at the time of activation of this instruction.
[0063] This invention is an information processing circuit containing the stack pointer assigned to two or more registers which were able to be continuously set in order, and one of general-purpose registers. The object code of one [at least] instruction of the continuation push instruction which has two or more register specific information in object code, and a continuation pop instruction is decoded. When at least one side of a means to output a control signal based on this object code, and said continuation push instruction and said continuation pop instruction is performed, At least one side of the processing which carries out multiple-times pop succeeding said two or more registers from the processing which carries out a multiple-times push from said two or more registers continuously to the stack prepared in memory, and said stack It is characterized by including the means performed based on the memory address specified according to the contents of said control signal and said stack-pointer dedicated register, and said two or more register specific information.

[0064] This invention relates to said continuation push instruction and said continuation pop instruction in the case of using a general-purpose register as a stack pointer.
[0065] According to this invention, if said continuation push instruction and said continuation pop instruction are executed, the same effectiveness as the case where multiple-times activation of the case where multiple-times activation of the push instruction is carried out continuously, or the pop instruction is carried out continuously will be acquired. That is, renewal of the stack pointer accompanying an exchange and this exchange of the data between two or more registers and a stack can be performed with one instruction. For this reason, when performing the data transfer between two or more registers and a stack, it can prevent object code size increasing by repeating a push instruction or a pop instruction. Moreover, improvement in the processing speed of interruption processing and a subroutine call return can be aimed at, without avoiding that a program execution step becomes long and consuming a useless cycle.
[0066] It is characterized by the information processing circuit of ******** being a RISC method.
[0067] The information processing circuit of a RISC method is designed for the purpose of miniaturizing hardware and attaining improvement in the speed. For this reason, it has many general-purpose registers and reduction of the number of instructions is aimed at by extracting an instruction set to the high thing of versatility.
[0068] Therefore, in the information processing circuit of a RISC method, the stack pointer was assigned to the general-purpose register, and when a stack pointer was treated, it was processing using the instruction set treating a general-purpose register. However, by such approach, an instruction length becomes large, and the utilization ratio of memory is not good.
[0069] According to this invention, in the information processing circuit of a RISC method, instruction lengths can be reduced and the utilization ratio of memory can be gathered.
[0070] The information processing circuit of this invention decodes a fixed-length instruction, and is characterized by performing executive operation based on this instruction.
[0071] If a fixed-length instruction is used, the time amount which decoding of an instruction takes can be shortened compared with the case where a variable-length instruction is used, and the circuit scale of an information processing circuit can be made small. When adopting a fixed-length instruction, in order to prevent making a redundant part to an instruction and to use memory efficiently, the number of bits required for each instruction has little dispersion, and its shorter possible one is desirable.
[0072] According to this invention, generally, an instruction length can shorten the instruction length of an instruction which deals with the stack pointer which tends to become long. Therefore, even if it is the case where a fixed-length instruction is adopted, it can prevent making a redundant part to an instruction, and memory can be used efficiently.
[0073] The microcomputer of this invention is characterized by including a means to perform I/O with the information processing circuit of this invention and storage means which were mentioned above, and the exterior.
[0074] According to this invention, processing speed can offer a quick microcomputer with the sufficient utilization ratio of memory.
[0075] The microcomputer of this invention is characterized by performing the program of the language which has the structure where relate with said stack pointer and the storage region of an auto variable is secured.
[0076] There is C as language which has the structure where relate with a stack pointer and the storage region of an auto variable is secured and to carry out. When the microcomputer of this

invention processes the program of such language, processing speed and the utilization ratio of memory can be raised effectively.
[0077] The electronic equipment of this invention is characterized by including the microcomputer of this invention mentioned above.
[0078] According to this invention, since processing speed builds in the quick good information processing circuit of the utilization ratio of memory, cheap and highly efficient electronic equipment can be offered.
[0079]
[Embodiment of the Invention] Hereafter, this example is explained based on a drawing.
[0080] (Example 1)
(1) By the architecture of a load store mold, CPU of configuration this example of CPU of this example executes almost all instructions in 1 cycle with a pipeline. All instructions are described by the 16-bit fixed length, and the instruction which CPU of this example processes has realized very small object code size.
[0081] Especially CPU of this example is constituted so that the instruction set of the instruction group only for stack pointers which has a register only for stack pointers in order to describe efficiently the processing which deals with a stack pointer and to perform it, and has the object code which makes this stack-pointer dedicated register an implied operand can be decoded and performed.
[0082] Drawing 1 is drawing for explaining the outline of the circuitry of CPU of this example.
[0083] A book CPU 10 contains a register set including SP14 who are a general-purpose register 11, PC12 with which the program counter is stored, the processor status register (PSR) 13, and a register only for stack pointers, an instruction decoder 20 and the immediate generation machine 22, the address adder 30, ALU40, the PC incrementer 44 and the various internal buses 72, 74, 76, and 78, the various internal signal lines 82, 84, and 86, and 88 grades.
[0084] Said instruction decoder 20 decodes the inputted object code, performs processing required in order to execute this instruction, and outputs a required control signal. In addition, this instruction decoder 20 functions also as said decode means to decode the object code of the instruction only for said stack pointers, and to output a control signal based on this instruction.
[0085] The immediate generation machine 22 generates the immediate data of 32 bits used at the time of activation based on the immediate contained in object code, or generates the constant data of 0 [required for activation of each instruction], **1, **2, and 4 [**]. The PC incrementer 44 updates the program counter stored in PC12 based on the execution cycle of an instruction. In case the address adder 30 adds the immediate data generated with the information stored in various registers, or the immediate generation vessel 22 and reads data from memory, it generates required address data. ALU40 performs math processing and logical operation.
[0086] Moreover, this CPU contains various buses and a signal line inside. PA_BUS72 and PB_BUS74 have the function to transmit the input signal of ALU40 etc. WW_BUS76 has the function which takes out the result of an operation of ALU40, and is transmitted to a general-purpose register. XA_BUS78 has the function to transmit the address data taken out from the general-purpose register 11 or SP14 etc. The IA signal line 82 transmits address data to external I_ADDR_BUS92 from each part inside CPU. The DA signal line 84 transmits address data to external D_ADDR_BUS96 from each part inside CPU. The DIN signal line 86 transmits data to each part inside CPU from D_DATA_BUS98 of the CPU exterior. The DOUT signal line 88 transmits data to external D_DATA_BUS98 from each part inside CPU. IA input change-over 83 switches the various signals (PA_BUS72, WW_BUS76, PC12, PC+2) outputted to the IA signal

line 82. The DOUT input change-over 89 switches the various signals (PA_BUS72, WW_BUS76, PC12, PC+2) outputted to the DOUT signal line 88.
[0087] Moreover, said each part of CPU10 functions based on the control signal which said instruction decoder 20 outputs also as said activation means to execute an instruction and to perform the instruction group only for stack pointers based on the contents of said control signal and said stack-pointer dedicated register.
[0088] A book CPU 10 performs an exchange of the exterior and a signal through the instruction address bus (I_ADDR_BUS) 92 for the 16-bit instruction data bus (I_DATA_BUS) 94 and an instruction data access, the 32-bit data bus (D_DATA_BUS) 98, the data address bus (D_ADDR_BUS) 96 for a data access, and the control bus that is not illustrated for a control signal.
[0089] (2) the explanation of a register set in which this example carries out CPU ** -- explain a part required about the outline of the register set which CPU of this example has next.
[0090] The register set which CPU of this example has in drawing 2 is shown. CPU of this example has the register set which contains 16, and PC12, PSR13, SP14, ALR (arithmetic operation low register)15 that is not illustrated and AHR (arithmetic operation high register)16 which is not illustrated for a general-purpose register 11.
[0091] said general-purpose register 11 -- functional -- a 32-bit equivalent register -- it is -- R0 to R15, and naming ******. This general-purpose register 11 is used at the time of a data operation and address computation.
[0092] Moreover, PC12 is the incremental counter of 32 bit length, and holds the program counter which is the address of the instruction under current activation. In the text, when pointing out a register name, it is called PC, and it is called a program counter when putting the value stored in PC.
[0093] Direct access of this PC12 cannot be carried out by a load instruction etc. If a call instruction, an int instruction, interruption, and an exception occur, a program counter will be read from PC12 and will be evacuated to a stack. Thus, it will fly, if branch instruction is executed, and the point address is set as PC. It is also the same as when branching with a conditional-branching instruction. And the return point instruction address is read by a ret instruction and reti instruction from a stack, and returns to PC12 with them.
[0094] PSR (processor status register)13 is a 32-bit register to which the flag is assigned, and holds the current condition of CPU. If an int instruction, interruption, an exception, etc. occur, in case it will branch to each manipulation routine, the condition of PSR at that time is evacuated to a stack. Conversely, by activation of a reti instruction, the evacuated value returns to PSR.
[0095] SP14 is a register only for [of 32 bits] stack pointers, and the stack pointer which puts the head address of a stack is stored. In the text, when pointing out a register name, it is called SP, and it is called a stack pointer when putting the value stored in SP. However, since the stack pointer has always pointed out the boundary of WORD, 2 bits of the low order of said stack pointer are always 0.
[0096] This stack pointer is updated with activation of the various instructions included in the instruction group only for stack pointers currently prepared by this example, and generating of a trap. There are an instruction which branches to other routines, such as a call instruction and a ret instruction, as an instruction only for stack pointers which updates a stack pointer, a stack-pointer migration instruction, a pushn instruction, a popn instruction, etc. For example, if a call instruction is executed, the decrement (-4) of the stack pointer will be first carried out only for wordsize (4), and PC12 will be evacuated to a stack. Moreover, if a ret instruction is executed,

the return point address will be conversely loaded to PC from a stack, and the wordsize part increment (+4) of the stack pointer will be carried out. When an int instruction is executed, or it interrupts and an exception etc. occurs, the value of PC or PSR is evacuated to a stack in the following procedures.
[0097] 1) Evacuate PC to the head address of the stack to which it is pointed out with SP=SP-42 stack pointer.
[0098] 3) Evacuate PSR to the head address of the stack to which it is pointed out with SP=SP-44 stack pointer.
[0099] If a reti instruction is executed, processing contrary to the above will be performed and CPU will return to a former condition. Thus, activation of a call instruction, a ret instruction, and an int instruction updates a stack pointer based on this activation. About the detail of each instruction, it mentions later.
[0100] Moreover, in this example, a trap means what doubled the exception generated by activation of interruption and the instruction which are generated in activation of an instruction asynchronous. If a trap occurs, CPU will read a vector table from a trap table, after evacuating a program counter (PC) and a processor status register (PSR) to a stack, and will branch to the manipulation routine corresponding to the trap. With generating of a trap, IE bit in PSR (enable [interruption]) is cleared, and generating of mask possible interruption beyond it is forbidden. In order to enable again external interruption in which a mask is possible, 1 is written in and set as IE bit of PSR using the load instruction to PSR.
[0101] A reti instruction is used in order to return to the original routine from trap processing. If a reti instruction is executed, CPU will branch to the return address while it is read from a stack in order of PSR and PC and returns PSR to the original value. In addition, there are a debugging exception, an address irregular train exception, an overflow exception, a division-by-zero exception, etc. as exception.
[0102] Explanation is omitted about ALR (arithmetic operation low register)15 and AHR (arithmetic operation high register)16.
[0103] The special registers PSR13, SP14, ALR15, and AHR16 which CPU has can perform data transfer between general-purpose registers using a load instruction. Each register has a special register number and is accessed using this number.
[0104]
A special register name Special register number The description approach processor status register of an assembler 0 % PSRSP 1 % SP arithmetic operation low register 2 % An ALR arithmetic operation high register From the lower part of the field which followed memory in the already kicked temporary storage, data accumulate a 3 %AHR (3) stack and the explanation stack about a stack pointer on ledged, and they are memorized. The stack pointer shows the address of the data memorized, the top, i.e., last, of a stack memory.
[0105] General actuation of a stack pointer is explained using drawing 3 .
[0106] 100 of drawing 3 expresses the stack area established in memory. Supposing a shadow area 102 is data stored at the end, the stack pointer stored in SP14 shows the memory address 1000 of these data. In addition, the lower part field 104 of a shadow area 102 is a field where data are already stored, and the upper part field 106 of a shadow area 102 shows the field where data will be stored from now on.
[0107] Since the stack pointer has always pointed out the boundary of WORD, when writing information in a stack, it stores information in the location where only 4 moves upwards the stack pointer with which SP14 was stored, and this stack pointer points to it. Moreover, when

taking out the information stored in the stack, the information on the address which current [SP / 14] shows is taken out, and only 4 moves downward the stack pointer stored in SP14. Thus, the stack pointer shows the storing address of the information always stored in the stack finally.
[0108] (4) Although a general-purpose register is used as a stack pointer in CPU of the explanation usual RISC method about the instruction group only for stack pointers, in this example, it has SP14 who is a register only for stack pointers, and is set as the actuation object of said instruction group only for stack pointers.
[0109] Said instruction group only for stack pointers is the generic name of two or more instructions which make an implied operand SP14 who is a register only for stack pointers, and operate it by being with SP14. This instruction group only for stack pointers includes the instruction which branches to other routines, such as SP relative load instructions (ld etc.), a stack-pointer migration instruction (add, sub), instructions (call etc.) that branch to a subroutine, and return instructions (ret etc.), a continuation push instruction (pushn), and a continuation pop instruction (popn).
[0110] Since it is the instruction only for stack pointers, I hear that that it is common in the instruction of ******** has the unnecessary information for specifying a stack pointer to object code, and it is in it. Therefore, the effectiveness that a short instruction length can describe the processing using a stack pointer efficiently is also common.
[0111] Moreover, if these instructions are used, information memorized by the stack prepared in memory can be processed efficiently. Moreover, interruption processing and processing of a subroutine call return can be performed efficiently.
[0112] Here, the busy condition of the already kicked stack and the condition of a stack pointer are explained to the example of use and memory of the instruction only for said stack pointers in case a subroutine is called using drawing 4 . The main program 500 and subroutine 520 of drawing 4 are the program described by the object code which the C compiler created. 540 shows the condition of the stack on memory. 502 shows that processing which uses general-purpose registers R0-R3 is performed. 506 shows the subroutine call instruction. The back stack pointer (SP) with which the instruction before the subroutine call instruction with which a subroutine call instruction is executed by the main program 500 (i.e., before being shown in 504) was executed presupposes that the address of ** on the stack 540 on memory was pointed out. If said subroutine call instruction is executed, control of activation will be crossed to a subroutine 520. At this time, the instruction (call instruction) which branches to the subroutine which is the instruction only for said stack pointers is executed by this example. As for the value of a stack pointer (SP), activation of this instruction stores the return address to a main program in the area 544 on the stack to which the increment only of -4 was carried out (refer to **SP of drawing 4), and this stack-pointer **SP has pointed it out automatically.
[0113] And in a subroutine 520 side, processing which transmits to a stack the value stored in the general-purpose registers R0-R3 currently used for the beginning of activation by the main program is performed. 524 shows the continuation push instruction (pushn) which is an instruction only for stack pointers which performs processing which shunts to a stack the value stored in general-purpose registers R0-R3. If this instruction is executed, the value stored in general-purpose registers R0-R3 will be continuously transmitted to a stack, and it is stored in a stack 540 as shown in 550 of drawing 4 . The stack pointer (SP) has pointed out 546 as activation of this processing finishes (524**) (**SP).
[0114] Next, in a subroutine 520, the auto variable area used by the subroutine is secured. The add instruction shown in 526 is a stack-pointer migration instruction which is an instruction only

for stack pointers, and secures the stack area which is made to move a stack pointer up and is used by the subroutine 520. If this instruction is executed, only X cutting tool will move a stack pointer (SP) to the upper location 548 (**SP), and the field of the auto variable which the subroutine shown in notes 2 uses will be secured.
[0115] It is shown that 528 performs processing which used an auto variable and general-purpose registers R0-R3 by the subroutine 520. At this time, the location of a stack pointer has pointed out **SP, and loading of an auto variable is performed by being with the load instruction only for SPs which is an instruction only for stack pointers.
[0116] 529 shows said SP load instruction which transmits the auto variable S1 stored on memory to a general-purpose register R1. Said auto variable S1 is stored in the location which has Y bytes of offset value from a stack pointer (**SP). Since a stack pointer does not move in the midst of processing of 528 of a subroutine as mentioned above, the memory address of an auto variable is specified with a stack-pointer + offset value, and can perform the exchange with a general-purpose register efficiently using the load instruction only for said SPs.
[0117] Moreover, before returning from a subroutine 520 to a main program 500, the value of the general-purpose registers R0-R3 which had shunted to the stack is returned to general-purpose registers R0-R3, and it must be made for a stack pointer to have to point out the area 544 where the return address to a main routine is stored. Therefore, the stack pointer to which it was made to move with the stack-pointer migration instruction of 526 is first returned to the location of a basis. The sub instruction shown in 530 is a stack-pointer migration instruction which is an instruction only for stack pointers, and moves a stack pointer caudad. Activation of this instruction moves a stack pointer to 546 (refer to **SP). Next, processing which restores the information 550 stored in the stack to general-purpose registers R0-R3 is performed. 532 shows the continuation pop instruction (popn) which is an instruction only for stack pointers which performs processing which transmits the information on a stack to general-purpose registers R0-R3. If this instruction is executed, the value 550 stored in the stack will be continuously transmitted to general-purpose registers R0-R3, and it is stored in a stack 540 as shown in 550 of drawing 4 . activation of this processing -- finishing (532**) -- the stack pointer has pointed out 544 (refer to **SP).
[0118] 534 shows the return instruction. If said return instruction is executed, control of activation will be crossed to a main program 500. At this time, the return instruction (ret instruction) which is an instruction only for said stack pointers is executed by this example. If this instruction is executed, it will branch to the instruction which the return address to the main program stored in the stack area which **SP points out shows, namely, will return to the instruction 507 next to a main program 500. And the value of a stack pointer moves to the head area of a stack where the increment only of +4 is carried out, and a main program 500 uses it automatically (refer to **SP of drawing 4).
[0119] The circuitry for executing explanation and this instruction of an instruction about each [said] instruction only for stack pointers below, respectively, the motion at the time of activation, etc. are explained to a detail.
[0120] (5) Create the object code a C compiler relates the field of an auto variable with a stack pointer, and it is remembered that carried out the stack-pointer relative load instruction above-mentioned. Specifically, the C compiler is the specification which secures Y bytes of field of an auto variable from a stack pointer in the place whose offset is X.
[0121] Drawing 5 is drawing for explaining the processing which transmits the auto variable on memory to a register. The auto variable a is [halfword data and auto variable d-g of WORD data

and the auto variables b and c] cutting tool data. The stack pointer stored in SP shows 1000 which is the address of the head field of the field 600 where the auto variable on a stack is stored. The area on the stack which makes said stack pointer a memory address as area for storing the auto variable a The area on the stack which makes a memory address a stack pointer +2 and a stack pointer +4 as area for storing the auto variables b and c, respectively The area on the stack which makes a memory address a stack pointer +5, a stack pointer +6, a stack pointer +7, and a stack pointer +8 as area for storing auto variable d-g, respectively is secured. And in case given processing is performed, data transfer processing is needed between the auto variable specified by the memory address of a stack-pointer + offset value, and a general-purpose register.
[0122] In CPU of this example, in order for short object code to describe such transfer processing and to perform it efficiently, the instruction set shown below is prepared as an SP relative load instruction which is one of the instructions only for stack pointers.
[0123]
ld.b %Rd [%sp+imm6] -- (1)
ld.ub %Rd [%sp+imm6] -- (2)
ld.h %Rd [%sp+imm7] -- (3)
ld.uh %Rd [%sp+imm7] -- (4)
ld.w %Rd [%sp+imm8] -- (5)
ld.b [%sp+imm6] %Rs -- (6)
ld.h [%sp+imm7] %Rs -- (7)
ld.w [%sp+imm8] %Rs -- (8)
(1) - (8) describes instruction code by the assembler. (1) is an instruction which carries out the sign escape of the cutting tool data from a stack, and is transmitted to a register. (2) is an instruction which carries out the zero escape of the cutting tool data from a stack, and is transmitted to a register. (3) is an instruction which carries out the sign escape of the halfword data from a stack, and is transmitted to a register. (4) is an instruction which carries out the zero escape of the halfword data from a stack, and is transmitted to a register. (5) is an instruction which transmits WORD data to a register from a stack. (6) is an instruction which transmits cutting tool data to a stack from a register, (7) is an instruction which transmits halfword data to a stack from a register, and (8) is an instruction which transmits WORD data to a stack from a register.
[0124] [%sp+imm6], [%sp+imm7], and [%sp+imm8] express immediate offset information, respectively, the offset value generated based on this immediate offset information at the time of activation and the value of the stack pointer stored in SP14 are added, and a memory address is generated. When the data size of the data on memory is a cutting tool, [%sp+imm7 of [%sp+imm6]] is immediate offset information when the data size of the data on memory is a halfword, in case the data size of the data on memory of [%sp+imm8] is WORD. These all <A mentioned later
HREF="/Tokujitu/tjitemdrw.ipdl?N0000=237&N0500=1E_N/;>??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;; ??6>; ; ??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;??6>;;

DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]
[Drawing 1] It is drawing for explaining the outline of the circuitry of CPU of this example.
[Drawing 2] The register set which CPU of this example has is shown.
[Drawing 3] It is drawing for explaining general actuation of a stack pointer.
[Drawing 4] It is drawing for explaining the busy condition of the stack already kicked by the example of use and memory of the instruction only for stack pointers, and the condition of a stack pointer.
[Drawing 5] It is drawing for explaining the processing which transmits the auto variable on memory to a register.
[Drawing 6] Drawing 6 (A) and (B) are drawings having shown the bit field of SP relative load instruction and a general-purpose load instruction.
[Drawing 7] It is a flow chart Fig. for explaining actuation of SP relative load instruction of WORD data read-out.
[Drawing 8] It is a flow chart Fig. for explaining actuation of SP relative load instruction of WORD data writing.
[Drawing 9] Drawing 9 (A) - (F) is drawing for explaining the busy condition of the stack on the memory by each routine in case a program is performed over two or more routines, and the condition of a stack pointer.
[Drawing 10] Drawing 10 (A) and (B) are drawings having shown the bit field of a stack-pointer migration instruction and general-purpose operation instruction.
[Drawing 11] It is a flow chart Fig. for explaining actuation of an addition stack-pointer migration instruction.
[Drawing 12] It is a flow chart Fig. for explaining actuation of a subtraction stack-pointer migration instruction.
[Drawing 13] It is drawing for explaining control of the program execution by a call instruction and ret instruction.
[Drawing 14] It is drawing having shown the bit field of PC relative subroutine call instruction.
[Drawing 15] It is a flow chart Fig. for explaining actuation of PC relative subroutine call instruction.
[Drawing 16] It is a flow chart Fig. for explaining actuation of a ret instruction.
[Drawing 17] Drawing 17 (A) and (B) are drawings having shown typically the motion at the time of a push instruction execution.
[Drawing 18] Drawing 18 (A) and (B) are drawings having shown typically the motion at the time of a pop instruction execution.
[Drawing 19] The bit map of pushn and a popn instruction is shown.
[Drawing 20] It is a block diagram for explaining the hardware configuration for executing a continuation push instruction (pushn) and a continuation pop instruction (popn).
[Drawing 21] It is a flow chart Fig. for explaining actuation of a pushn instruction.
[Drawing 22] It is a flow chart Fig. for explaining actuation of a popn instruction.
[Drawing 23] It is the hardware block diagram of the microcomputer of the gestalt of this operation.
[Description of Notations]
2 Microcomputer

10 CPU
11 General-purpose Register
12 PC (Program Counter)
13 PSR (Processor Status Register)
20 Instruction Decoder
22 Immediate Generation Machine Term
24 Offset Signal
30 Address Adder
32 Latch (Add_LT)
40 ALU
44 PC Incrementer
45 Control Circuit Block
46 Four Bit Counters (Countx)
50 Memory (RAM)
52 Memory (ROM)
54 Register-Select Address Signal
60 Bus Control Unit (BCU)
72, 74, 76, 78 Internal bus
82, 84, 86, 88 Internal signal line
92, 94, 96, 98 External bus

CORRECTION OR AMENDMENT

[Kind of official gazette] Printing of amendment by the convention of 2 of Article 17 of Patent Law
[Section partition] The 3rd partition of the 6th section
[Publication date] March 14, Heisei 15 (2003. 3.14)

[Publication No.] JP,10-91443,A
[Date of Publication] April 10, Heisei 10 (1998. 4.10)
[Annual volume number] Open patent official report 10-915
[Application number] Japanese Patent Application No. 9-135923
[The 7th edition of International Patent Classification]
G06F 9/42 330
9/30 350
9/46 313
[FI]
G06F 9/42 330 A
9/30 350 B
9/46 313 Z

[Procedure revision]
[Filing Date] December 12, Heisei 14 (2002. 12.12)
[Procedure amendment 1]
[Document to be Amended] Specification
[Item(s) to be Amended] The name of invention
[Method of Amendment] Modification
[Proposed Amendment]
[Title of the Invention] An information processing circuit and a microcomputer
[Procedure amendment 2]
[Document to be Amended] Specification
[Item(s) to be Amended] Claim
[Method of Amendment] Modification
[Proposed Amendment]
[Claim(s)]
[Claim 1] The stack-pointer dedicated register only used for stack pointers,
A decode means to have the object code which makes this stack-pointer dedicated register an implied operand, to decode the object code of the instruction group only for stack pointers the processing based on this stack-pointer dedicated register was described to be, and to output a control signal based on this object code,
The information processing circuit characterized by including an activation means to perform said instruction group only for stack pointers based on the contents of said control signal and said stack-pointer dedicated register.
[Claim 2] In claim 1,
Said instruction group only for stack pointers includes the load instruction which has transfer register specific information in object code,
Said decode means
Said load instruction is decoded,
Said activation means
The information processing circuit characterized by performing either [at least] the data transfer from the first area to the first given given register on memory, or the data transfer from said first given register to said first given area based on the register address specified by the memory address specified with said stack-pointer dedicated register, and said transfer register specific information in case said load instruction is executed.
[Claim 3] In either claim 1 - claim 2,
A stack-pointer migration instruction for said instruction group only for stack pointers to have migration information in object code, and move a stack pointer to it is included,
Said decode means
Said stack-pointer migration instruction is decoded,
Said activation means
The information processing circuit characterized by changing the contents of said stack-pointer dedicated register based on said migration information in case said stack-pointer migration instruction is executed.
[Claim 4] In either claim 1 - claim 3,
Two or more registers which were able to be continuously set in order are included,
Said instruction group only for stack pointers includes either [at least] the continuation push instruction which has two or more register specific information in object code, or a continuation

pop instruction,
Said decode means
Either [at least] said continuation push instruction or a continuation pop instruction is decoded,
Said instruction-execution means
The information-processing circuit characterized by to attain to the memory address specified according to the contents of said stack-pointer dedicated register, and to perform at least one side of the processing which carries out multiple-times pop succeeding two or more of said registers from the processing which carries out a multiple-times push from two or more of said registers continuously to the stack prepared in memory, and said stack based on said two or more register specific information in case either [at least] said continuation push instruction or said continuation pop instruction is executed.
[Claim 5] In either claim 1 - claim 4,
The program counter register only for program counters is included,
Said instruction group only for stack pointers includes the branch instruction which is the instruction which branches to a subroutine, and a return instruction from said subroutine,
Said decode means
Said branch instruction is decoded,
Said instruction-execution means
A means to perform at least one side of the return to shunting of the contents of said program counter register to the second given area of the stack prepared in memory, and the program counter register of the contents of said second area based on the memory address specified with said stack-pointer dedicated register in case said branch instruction is executed,
The information processing circuit characterized by including a means to update the contents of said stack-pointer dedicated register based on said shunting and said return.
[Claim 6] It is an information processing circuit containing the stack pointer assigned to two or more registers which were able to be continuously set in order, and one of general-purpose registers,
A means to decode the object code of one [at least] instruction of the continuation push instruction which has two or more register specific information in object code, and a continuation pop instruction, and to output a control signal based on this object code,
The information-processing circuit characterized by to include a means perform at least one side of the processing which carries out multiple-times pop succeeding two or more of said registers from the processing which carries out a multiple-times push from two or more of said registers continuously to the stack prepared in memory, and said stack based on the memory address specified according to the contents of said control signal and said stack-pointer dedicated register, and said two or more register specific information in case either [at least] said continuation push instruction or said continuation pop instruction execute.
[Claim 7] In either claim 1 - claim 6,
The information processing circuit characterized by being a RISC method.
[Claim 8] In either claim 1 - claim 7,
The information processing circuit characterized by decoding a fixed-length instruction and performing executive operation based on this instruction.
[Claim 9] The microcomputer characterized by including a means to perform I/O with an information processing circuit, a storage means, and the exterior according to claim 1 to 8.
[Procedure amendment 3]
[Document to be Amended] Specification

[Item(s) to be Amended] 0001
[Method of Amendment] Modification
[Proposed Amendment]
[0001]
[Field of the Invention] This invention relates to the microcomputer which contains an information processing circuit and said information processing circuit.

乙第14号証

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平10－91443

(43)公開日　平成10年(1998)４月10日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/42	３３０	Ｇ０６Ｆ	9/42	３３０Ａ
	9/30	３５０		9/30	３５０Ｂ
	9/46	３１３		9/46	３１３Ｚ

審査請求　未請求　請求項の数17　ＦＤ　（全 29 頁）

(21)出願番号　特願平9－135923

(22)出願日　平成９年(1997)５月８日

(31)優先権主張番号　特願平8－127541

(32)優先日　平８(1996)５月22日

(33)優先権主張国　日本（ＪＰ）

(71)出願人　000002369
セイコーエプソン株式会社
東京都新宿区西新宿２丁目４番１号

(72)発明者　久保田　哲
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

(72)発明者　工藤　真
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

(72)発明者　宮山　芳幸
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

(74)代理人　弁理士　井上　一　（外２名）

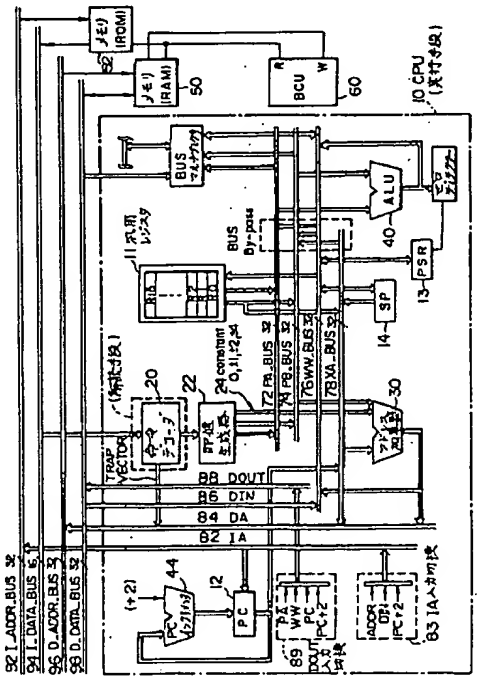
最終頁に続く

(54)【発明の名称】　情報処理回路、マイクロコンピュータ及び電子機器

(57)【要約】

【課題】　スタックポインタを取り扱う処理を、短い命令長で効率よく記述し、実行すること及びレジスタ退避やレジスタ復旧の処理を、効率よく記述し、割り込み処理及びサブルーチンコール・リターンの処理速度を向上させること。

【解決手段】　ＣＰＵ１０は、スタックポインタ専用のレジスタＳＰ１４を含みＳＰ１４を暗黙のオペランドとするスタックポインタ専用命令群を命令デコーダ２０で解読する。そして、汎用レジスタ１１、ＰＣ１２、ＳＰ１４、アドレス加算器３０、ＡＬＵ４０、ＰＣインクリメンタ４４、内部バス７２、７４、７６、７８、内部信号線８２、８４、８６、８８、外部バス９２、９４、９６、９８等を用いて前記スタックポインタ専用命令群をハードウエア的に実行する。前記スタックポインタ専用命令群は、ＳＰ相対ロード命令やスタックポインタ移動命令やｃａｌｌ命令やｒｅｔ命令や連続プッシュ命令や連続ポップ命令を含む。



1

【特許請求の範囲】
【請求項１】　スタックポインタ専用に用いるスタックポインタ専用レジスタと、
該スタックポインタ専用レジスタを暗黙のオペランドとするオブジェクトコードを有し、該スタックポインタ専用レジスタに基づく処理が記述されたスタックポインタ専用命令群のオブジェクトコードを解読して、該オブジェクトコードに基づき制御信号を出力する解読手段と、
前記スタックポインタ専用命令群を、前記制御信号及び前記スタックポインタ専用レジスタの内容に基づき実行する実行手段とを含むことを特徴とする情報処理回路。
【請求項２】　請求項１において、
前記スタックポインタ専用命令群が、転送レジスタ特定情報をオブジェクトコードに有するロード命令を含み、
前記解読手段が、
前記ロード命令を解読し、
前記実行手段が、
前記ロード命令を実行する際、メモリ上の所与の第一のエリアから所与の第一のレジスタへのデータの転送及び前記所与の第一のレジスタから前記所与の第一のエリアへのデータの転送の少なくとも一方を、前記スタックポインタ専用レジスタにより特定されるメモリアドレス及び前記転送レジスタ特定情報により特定されるレジスタアドレスに基づき行うことを特徴とする情報処理回路。
【請求項３】　請求項２において、
前記ロード命令が、前記メモリ上の前記第一のエリアのアドレスを特定するためのオフセットに関する情報であるオフセット情報をオブジェクトコードに含み、
前記実行手段が、
前記スタックポインタ専用レジスタの内容と前記オフセット情報により前記メモリアドレスを特定することを特徴とする情報処理回路。
【請求項４】　請求項３において、
前記オフセット情報が、即値で与えられた即値オフセット情報とメモリ上の所与のデータのサイズに関するデータサイズ情報とを含み、
前記実行手段が、
前記即値オフセット情報と前記データサイズ情報とに基づき、前記即値オフセット情報を左論理シフトしてオフセット値を生成し、前記スタックポインタ専用レジスタの内容と前記オフセット値を加算した値により前記メモリアドレスを特定することを特徴とする情報処理回路。
【請求項５】　請求項１～請求項４のいずれかにおいて、
前記スタックポインタ専用命令群が、オブジェクトコードに移動情報を有しスタックポインタを移動するためのスタックポインタ移動命令を含み、
前記解読手段が、
前記スタックポインタ移動命令を解読し、
前記実行手段が、

2

前記スタックポインタ移動命令を実行する際、前記移動情報に基づき、前記スタックポインタ専用レジスタの内容を変更することを特徴とする情報処理回路。
【請求項６】　請求項５において、
前記移動情報が、即値で与えられた即値移動情報を含み、
前記命令実行手段が、
前記即値移動情報と前記スタックポインタ専用レジスタの内容とを加算する処理及び前記スタックポインタ専用レジスタの内容から前記即値移動情報を減算する処理の少なくとも一方の処理を行うことを特徴とする情報処理回路。
【請求項７】　請求項１～請求項６のいずれかにおいて、
連続して順序づけられた複数のレジスタを含み、
前記スタックポインタ専用命令群が、複数レジスタ特定情報をオブジェクトコードに有する連続プッシュ命令及び連続ポップ命令の少なくとも一方を含み、
前記解読手段が、
前記連続プッシュ命令及び連続ポップ命令の少なくとも一方を解読し、
前記命令実行手段が、
前記連続プッシュ命令及び前記連続ポップ命令の少なくとも一方を実行する際、前記複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記スタックポインタ専用レジスタの内容により特定されるメモリアドレスに及び前記複数レジスタ特定情報とに基づき行うことを特徴とする情報処理回路。
【請求項８】　請求項７において、
０からｎ－１までのレジスタ番号で特定されたｎ個の汎用レジスタを含み、
前記連続プッシュ命令及び連続ポップ命令の少なくとも一方のオブジェクトコードが、前記複数レジスタ特定情報として、前記レジスタ番号のいずれかが指定された最終レジスタ番号を含み、
前記実行手段が、
レジスタ０から前記最終レジスタ番号で特定されるレジスタまでの複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記スタックポインタ専用レジスタの内容により特定されるメモリアドレスに基づき行うことを特徴とする情報処理回路。
【請求項９】　請求項７又は８のいずれかにおいて、
前記実行手段が、
前記複数のレジスタのいずれか所与のレジスタの内容を、前記スタックポインタ専用レジスタで特定されるメモリアドレスに基づきメモリに設けられたスタックに書

き込む書き込み手段と、
前記書き込み手段による前記スタックへの書き込み回数をカウントする書き込み回数カウント手段と、
前記カウント手段によってカウントされた前記書き込み回数と前記複数レジスタ特定情報の値を比較する比較手段とを含み、
前記書き込み手段が、
第一の入力と第二の入力を加算器で加算し書き込み先を特定するための書き込みメモリアドレスを生成する書き込みメモリアドレス生成手段と、
前記加算器の第一の入力が、連続専用命令の実行開始時にはスタックポインタ専用レジスタの内容となるように制御し、それ以降は書き込みアドレス生成手段によって生成された書き込みアドレスとなるよう制御する第一の入力制御手段と、
前記加算器の第二の入力に前記スタックから１ワードを書き込んだときのオフセット値を出力する第二の入力制御手段と、
前記複数レジスタ特定情報から前記書き込み回数を減じた値で特定されるレジスタの内容を、前記書き込みメモリアドレスに基づき前記スタックに書き込む手段とを含み、
前記比較手段の比較結果に基づき、複数のレジスタからの前記スタックへの書き込み及び書き込み終了を制御することを特徴とする情報処理回路。
【請求項１０】　請求項７～９のいずれかにおいて、
前記命令実行手段が、
前記スタックポインタ専用レジスタで特定されるメモリアドレスに基づきメモリに設けられたスタックの内容を読み出し、前記複数のレジスタのいずれか所与のレジスタに格納する読み出し手段と、
前記読み出し手段による前記スタックからの読み出し回数をカウントする読み出し回数カウント手段と、
前記カウント手段によってカウントされた前記読み出し回数と前記複数レジスタ特定情報の値を比較する比較手段とを含み、
前記読み出し手段が、
第一の入力と第二の入力を加算器で加算し書き込み先を特定するための書き込みメモリアドレスを生成する書き込みメモリアドレス生成手段と、
前記加算器の第一の入力が、連続専用命令の実行開始時にはスタックポインタ専用レジスタの内容となるように制御し、それ以降は読み出しアドレス生成手段によって生成された読み出しアドレスとなるよう制御する第一の入力制御手段と、
前記加算器の第二の入力に前記スタックから１ワードを書き込んだときのオフセット値を出力する第二の入力制御手段と、
前記読み出しメモリアドレスに基づき前記スタックの内容を読み出し、前記書き込み回数に基づき特定されるレジスタに格納する読み出し手段とを含み、

前記比較手段の比較結果に基づき、前記スタックの内容の読み出し及び読み出し終了を制御することを特徴とする情報処理回路。
【請求項１１】　請求項１～請求項１０のいずれかにおいて、
プログラムカウンタ専用のプログラムカウンタレジスタを含み、
前記スタックポインタ専用命令群が、サブルーチンへ分岐する命令及び前記サブルーチンからのリターン命令である分岐命令を含み、
前記解読手段が、
前記分岐命令を解読し、
前記命令実行手段が、
前記分岐命令を実行する際、メモリに設けられたスタックの所与の第二のエリアへの前記プログラムカウンタレジスタの内容の待避及び前記第二のエリアの内容のプログラムカウンタレジスタへの復帰の少なくとも一方を、前記スタックポインタ専用レジスタにより特定されるメモリアドレスに基づき行う手段と、
前記待避及び前記復帰に基づき前記スタックポインタ専用レジスタの内容を更新する手段とを含むことを特徴とする情報処理回路。
【請求項１２】　連続して順序づけられた複数のレジスタと、いずれかの汎用レジスタに割り当てられたスタックポインタを含む情報処理回路であって、
複数レジスタ特定情報をオブジェクトコードに有する連続プッシュ命令及び連続ポップ命令の少なくとも一方の命令のオブジェクトコードを解読して、該オブジェクトコードに基づき制御信号を出力する手段と、
前記連続プッシュ命令及び前記連続ポップ命令の少なくとも一方を実行する際、前記複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記制御信号及び前記スタックポインタ専用レジスタの内容により特定されるメモリアドレス及び前記複数レジスタ特定情報とに基づき行う手段を含むことを特徴とする情報処理回路。
【請求項１３】　請求項１～請求項１２のいずれかにおいて、ＲＩＳＣ方式であることを特徴とする情報処理回路。
【請求項１４】　請求項１～請求項１３のいずれかにおいて、
固定長の命令を解読し、該命令に基づき実行処理を行うことを特徴とする情報処理回路。
【請求項１５】　請求項１～請求項１４のいずれかに記載の情報処理回路と記憶手段と外部との入出力を行う手段とを含むことを特徴とするマイクロコンピュータ。
【請求項１６】　請求項１５において、

5

前記スタックポインタに関連づけてオート変数の記憶領域が確保される構造を有する言語のプログラムが実行されることを特徴とするマイクロコンピュータ。

【請求項１７】　請求項１５又は１６のいずれかに記載されたマイクロコンピュータを含むことを特徴とする電子機器。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、情報処理回路、前記情報処理回路を内蔵するマイクロコンピュータ、及び該マイクロコンピュータを用いて構成された電子機器に関する。

【０００２】

【背景技術及び発明が解決しようとする課題】従来、３２ビットのデータを処理できるＲＩＳＣ方式のマイクロコンピュータでは、３２ビット幅の固定長命令が用いられていた。その理由は、固定長命令を用いると、可変長命令を用いる場合に比べ、命令のデコードに要する時間を短縮でき、また、マイクロコンピュータの回路規模を小さくすることが出来るからである。

【０００３】ところが、３２ビットのマイクロコンピュータにおいても、特に３２ビットも必要としない命令も多い。従って全ての命令を３２ビットで記述すると、命令に冗長な部分が生じる命令が多くなり、メモリーの使用効率が悪くなる。

【０００４】そこで、本願の発明者は、制御回路を複雑にすることなくメモリの使用効率を向上させるために、処理出来るデータのビット幅より短いビット幅の固定長命令を処理するマイクロコンピュータについての検討を行っていた。

【０００５】しかし、例えば３２ビット固定長の命令を単に１６ビットの固定長にすると以下のような問題点が生じる。

【０００６】即ちＲＩＳＣ方式のマイクロコンピュータでは処理及び命令セットの汎用性を重視するため、スタックポインタを取り扱う処理を行う場合、いずれかの汎用レジスタをスタックポインタとして用い、汎用レジスタを操作対象とする命令を用いて前記処理を行っていた。従って、このような処理を記述する際の前記命令のには、スタックポインタとして使用している汎用レジスタの指定が必要となる。

【０００７】例えば、スタックポインタに所与のオフセット値を加えたメモリアドレスで特定されるデータを所与のレジスタに転送する処理を汎用レジスタを操作対象とする命令で記述する場合、その命令のオブジェクトコードには、前記オフセット値、前記所与のレジスタを特定する情報、スタックポインタとして使用しているレジスタを特定する情報が必要となる。

【０００８】このように、汎用レジスタを操作対象とする命令でスタックポインタを取り扱う処理を記述する場合、オブジェクトコードで指定する情報が多くなるため、命令の内容を１６ビットの固定長で記述することが困難となる。ここで命令長を例えば３２ビットにすると、命令の中にも特に命令に３２ビットも必要としない命令も多いため、命令に冗長な部分を生じる命令が多くなり、メモリの使用効率の悪化を招く。

6

【０００９】また、命令長が長くなればそれを格納するメモリも余分に必要になるため、メモリの効率的な使用という観点からは、固定長の命令に限らず、命令長を短くできることが好ましい。

【００１０】また、例えばＣ言語のようにスタックポインタに関連づけてオート変数の記憶領域が確保される言語で記述されたプログラムを実行させる場合、スタックポインタを取り扱う命令が多くなるため、スタックポインタを取り扱う命令を効率的に記述し、実行させることが望ましい。

【００１１】そこで、スタックポインタを取り扱う処理を行う場合、できるだけ短い命令長でその命令内容を記述し、実行することができるアーキテクチャが望まれていた。

【００１２】また最近の特にＲＩＳＣ方式のＣＰＵは、性能を高めるため内部に多くの汎用レジスタを持つようになった。多くのレジスタを内部に持つことによりメモリにアクセスすることなくＣＰＵ内部で高速に多くの処理を出来るからである。このように内部レジスタを多く持つと、割り込み処理やサブルーチンコールの時のレジスタ退避と復旧の処理の際、退避すべきレジスタ数が多くなる。

【００１３】以下、スタック系命令の中でもサブルーチンに入ったりでたりする際に多用するレジスタの待避、復帰命令を例にとり従来例を説明する。

【００１４】通常マイクロコンピュータの命令セットはＣＰＵのレジスタをメモリに設けたスタックに待避したり復旧するための命令を持っている。そのための専用の命令を持つもの、あるいはレジスタ間接アドレッシング命令をもつものがある。

【００１５】前記待避したり復旧したりするための命令に関する技術としては、インテル社の８０３８６に関して、「８０３８６プログラミン」（John H. Crawford Patric P. Gelsigner 著　岩谷宏　訳）に以下のような記述がある。

【００１６】即ちレジスタをスタックに書き込む命令として　ｐｕｓｈ、ｐｕｓｈａ、ｐｕｓｈｄがあり，スタックからレジスタにデータを戻すための命令としてｐｏｐ、ｐｏｐａ、ｐｏｐｄがある。

【００１７】ｐｕｓｈ命令でレジスタをスタックに書き込むときは　ｐｕｓｈ　ＥＡＸ　のようにオペランドとしてレジスタを指定する。これは３２ビットレジスタＥＡＸの場合である。レジスタＥＡＸ、ＥＣＸ、ＥＤＸ、ＥＢＸを全てスタックに書く場合は、

ｐｕｓｈ　ＥＡＸ
ｐｕｓｈ　ＥＣＸ
ｐｕｓｈ　ＥＤＸ
ｐｕｓｈ　ＥＢＸ
のようにｐｕｓｈ命令を繰り返す。

【００１８】このようにｐｕｓｈ、ｐｏｐ命令でレジスタを１本ずつ操作しているとオブジェクトコードのサイズは大きくなり、プログラムの実行ステップも多くなり、そのためプログラムの実行時間や処理動作はより遅くなる。

【００１９】そこで８０３８６の有する８本の汎用レジスタ全てをスタックに書き込むにはｐｕｓｈａ　または　ｐｕｓｈａｄ　命令を使う。ｐｕｓｈａは８本のレジスタの其々の下位１６ビットレジスタを対象とし、ｐｕｓｈａｄは３２ビットを対象とする。ｐｕｓｈａ、ｐｕｓｈａｄによりｐｕｓｈ命令を８回繰り返すことを省略できる。

【００２０】ｐｏｐ、ｐｏｐａ、ｐｏｐａｄ命令についても同様である。

【００２１】ｐｕｓｈ命令を繰り返すことの不利な点はプログラムコードが長くなること、１命令毎フェッチし実行するため実行が遅いことである。この点は、前記ｐｕｓｈａ、ｐｕｓｈａｄにより、前記８本のレジスタを全てスタックに書き込む場合には大きく改善される。しかし、４本の場合とか６本の場合等のように８未満のレジスタを書き込む場合には、この点は改善されない。

【００２２】即ち、８０３８６のようにｐｕｓｈａ、ｐｕｓｈａｄ、ｐｏｐａ、ｐｏｐａｄ命令で全てのレジスタを操作すると、レジスタの全てを退避・復旧する必要の無いときも、このサイクルの長い遅い命令を使わなければならない。このような場合、命令は一命令で済むがこの命令の実行サイクルは長くなるという問題が生じる。

【００２３】また、ｃａｌｌ命令やリターン命令等でサブルーチンへ分岐したり、呼ばれたルーチンへ戻る際には、戻り先アドレスとして必要となるプログラムカウンタの待避や復旧の処理が必要となる。従来のＲＩＳＣ方式のＣＰＵにおいては、これらの処理をソフトウエア的に実現していた。即ち、該処理を記述したアセンブラ命令（オブジェクトコード）を実行することにより、前記プログラムカウンタの待避や復旧の処理を行っていた。このため、ｃａｌｌ命令やリターン命令はオブジェクトコードの増加を招き、また１命令毎フェッチし実行するために実行速度の鈍化を招いていた。

【００２４】本発明の目的は、スタックポインタを取り扱う処理を、短い命令長で効率よく記述し、実行することができるアーキテクチャを有する情報処理回路、マイクロコンピュータ、電子機器を提供することである。

【００２５】また　本発明の他の目的は、レジスタ退避やレジスタ復旧の処理を、効率よく記述し、割り込み処理及びサブルーチンコール・リターンの処理速度が速い情報処理回路、マイクロコンピュータ、電子機器を提供することである。

【００２６】

【課題を解決するための手段】本発明は、スタックポインタ専用に用いるスタックポインタ専用レジスタと、該スタックポインタ専用レジスタを暗黙のオペランドとするオブジェクトコードを有し、該スタックポインタ専用レジスタに基づく処理が記述されたスタックポインタ専用命令群のオブジェクトコードを解読して、該オブジェクトコードに基づき制御信号を出力する解読手段と、前記スタックポインタ専用命令群を、前記制御信号及び前記スタックポインタ専用レジスタの内容に基づき実行する実行手段とを含むことを特徴とする。

【００２７】ここにおいてオブジェクトコードとは、一般に翻訳プログラムによって機械語に翻訳した結果得られるプログラムコードのことをいうが、本発明においては、翻訳プログラムによるかいなかに関係なく、機械語で記述されているプログラムコードを含む広い概念で使用する。

【００２８】本発明の情報処理回路は、スタックポインタ専用のスタックポインタ専用レジスタを有し、該スタックポインタ専用レジスタを操作対象とするスタックポインタ専用命令群の解読、実行を行うよう構成されている。

【００２９】前記スタックポインタ専用命令群は、スタックポインタ専用レジスタを取り扱うための専用のオペコードを有しているため、オブジェクトコードのオペランドにスタックポインタを特定するための情報を必要としない。言い換えれば、前記スタックポインタ専用命令群はスタックポインタ専用レジスタを暗黙のオペランドとしている。このため、汎用レジスタの１つをスタックポインタに割あて、汎用レジスタを操作対象とする命令を用いてスタックポインタの操作を行う場合に比べて、短い命令長でスタックポインタを取り扱う命令を記述することができる。

【００３０】従って本発明によれば、スタックポインタを取り扱う処理を短い命令長で記述し実行することができる情報処理回路を提供することができる。また、命令を記憶するメモリの使用効率のよい情報処理回路を提供することができる。

【００３１】本発明は、前記スタックポインタ専用命令群が、転送レジスタ特定情報をオブジェクトコードに有するロード命令を含み、前記解読手段が、前記ロード命令を解読し、前記実行手段が、前記ロード命令を実行する際、メモリ上の所与の第一のエリアから所与の第一のレジスタへのデータの転送及び前記所与の第一のレジスタから前記所与の第一のエリアへのデータの転送の少なくとも一方を、前記スタックポインタ専用レジスタにより特定されるメモリアドレス及び前記転送レジスタ特定

情報により特定されるレジスタアドレスに基づき行うことを特徴とする。

【００３２】ここにおいて前記スタックポインタ専用命令群に含まれる前記ロード命令とは、メモリとレジスタ間でデータの転送を行う命令であり、メモリからレジスタへの転送及びレジスタからメモリへの転送の少なくとも一方を含む概念である。なお、データの内容は問わずアドレスデータも含む概念である。メモリアドレスとは、転送の際のメモリ上のエリアを特定するためのアドレスをいう。

【００３３】前記ロード命令は、スタックポインタ専用レジスタを操作対象とするための専用のオペコードを有しているため、オブジェクトコードのオペランドにスタックポインタを特定するための情報を必要としない。このため、スタックポインタに関連づけたメモリアドレスを有するメモリ上のエリアとレジスタ間でデータの転送処理を行わせる場合、短い命令長で記述することができる。

【００３４】本発明は、前記ロード命令が、前記メモリ上の前記第一のエリアのアドレスを特定するためのオフセットに関する情報であるオフセット情報をオブジェクトコードに含み、前記実行手段が、前記スタックポインタ専用レジスタの内容と前記オフセット情報により前記メモリアドレスを特定することを特徴とする。

【００３５】ここにおいてオフセット情報とは、オフセット値を直接即値で指定したものでもよいし、オフセット値が格納されたレジスタ等のアドレスを指定する場合のように間接的に指定する場合でもよい。オフセット情報を含んだ前記ロード命令が実行される場合、転送の際に必要となる前記メモリアドレスは、スタックポインタ専用レジスタの内容及びオフセット情報に基づき特定される。

【００３６】従ってスタックポインタ及び前記オフセット情報に基づき特定されるメモリアドレスを有するメモリ上のエリアとレジスタ間でデータの転送処理を行わせる場合、短い命令長で記述することができる。

【００３７】また本発明によれば、例えばスタックポインタが常にワード境界をさすような構造の情報処理回路においても、適当なオフセット情報を指定することにより、スタック上の任意のエリアを指定することが可能となる。このため、データのサイズに応じて効率良くスタックに格納することができ、スタックの使用効率の向上を図ることができる。

【００３８】本発明は、前記オフセット情報が、即値で与えられた即値オフセット情報とメモリ上の所与のデータのサイズに関するデータサイズ情報とを含み、前記実行手段が、前記即値オフセット情報と前記データサイズ情報とに基づき、前記即値オフセット情報を左論理シフトしてオフセット値を生成し、前記スタックポインタ専用レジスタの内容と前記オフセット値を加算した値により前記メモリアドレスを特定することを特徴とする。

【００３９】ここにおいて即値オフセット情報とは、オフセット値を直接即値で指定したものをいう。またデータサイズ情報とは、転送すべきメモリ上データのサイズをいう。通常データサイズは、８ビットのバイトデータや、１６ビットのハーフワードデータ、３２ビットのワードデータ等の2^n（ｎは３以上）で表される。メモリ上のアドレスはバイト単位に与えられており、ハーフワードデータはハーフワード境界におかれ、ワードデータはワード境界におかれる。従って、ハーフワードデータのメモリアドレスの下位の１ビットは０となり、ワードデータのメモリアドレスの下位の２ビットは００となる。スタックポインタのアドレスは、ワード境界を指しているため、ハーフワードデータのメモリアドレスを生成する場合のオフセット値の下位の１ビットは０となり、ワードデータのメモリアドレスを生成する場合のオフセット値の下位の２ビットは００となる。

【００４０】また、左論理シフトとは、データのビット列を左にずらし、シフトによってデータの右側にでるあきビット（シフトインビット）に０が入るシフトをいう。

【００４１】本発明によれば、データサイズに基づき、前記オフセット即値情報の左論理シフトを行うため、データサイズによって一義的に決まる下位のビットを省略して、即値オフセット情報を記述することができる。従って、即値オフセット情報を効率的に指定することができ、バイト以外のデータサイズの場合、そのまま指定する場合に比べて、より大きいオフセット値に指定が可能となる。

【００４２】また該命令を用いることによって、データのメモリへの書き込み、読み出し時にそのデータサイズに応じた適切な境界位置が選択されることになる。

【００４３】本発明は、前記スタックポインタ専用命令群が、オブジェクトコードに移動情報を有しスタックポインタを移動するためのスタックポインタ移動命令を含み、前記解読手段が、前記スタックポインタ移動命令を解読し、前記実行手段が、前記スタックポインタ移動命令を実行する際、前記移動情報に基づき、前記スタックポインタ専用レジスタの内容を変更することを特徴とする。

【００４４】前記スタックポインタ移動命令は、スタックポインタ専用レジスタを操作対象とするための専用のオペコードを有しているため、オブジェクトコードのオペランドにスタックポインタを特定するための情報を必要としない。このため、スタックを移動させたい場合、短い命令長で記述することができる。従って、スタックに格納されているデータの処理やスタックポインタに関連づけて記憶されているデータの処理を行う際の命令の記述量を削減することができる。

【００４５】本発明によればスタックの移動が容易に行

えるため、特に異なるルーチンにおいてそれぞれ異なるスタックエリアを確保して処理を行う場合に有効である。即ちルーチンごとにスタックポインタを移動させる処理を行うことで、広範囲な領域にわたるアドレス指定が可能となる。

【００４６】本発明は、前記移動情報が、即値で与えられた即値移動情報を含み、前記命令実行手段が、前記即値移動情報と前記スタックポインタ専用レジスタの内容とを加算する処理及び前記スタックポインタ専用レジスタの内容から前記即値移動情報を減算する処理の少なくとも一方の処理を行うことを特徴とする。

【００４７】本発明によれば、スタックポインタの値を即値移動情報で指定された分だけ上方又は下方に移動させる処理を短い命令長で記述することができる。

【００４８】本発明は、連続して順序づけられた複数のレジスタを含み、前記スタックポインタ専用命令群が、複数レジスタ特定情報をオブジェクトコードに有する連続プッシュ命令及び連続ポップ命令の少なくとも一方を含み、前記解読手段が、前記連続プッシュ命令及び連続ポップ命令の少なくとも一方を解読し、前記命令実行手段が、前記連続プッシュ命令及び前記連続ポップ命令の少なくとも一方を実行する際、前記複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記スタックポインタ専用レジスタの内容により特定されるメモリアドレスに及び前記複数レジスタ特定情報とに基づき行うことを特徴とする。

【００４９】プッシュとはメモリに設けられたスタックにデータを積み重ねて格納すること、ポップとは前記スタックからデータを取り出すことをいう。ここにおいてプッシュする処理及びポップする処理とは、前記格納及び取り出しの処理と、それに伴うスタックポインタの更新処理を含む。通常の情報処理回路は１のレジスタからスタックにデータやアドレスを格納するためのｐｕｓｈ命令、スタックの内容をレジスタに取り出すｐｏｐ命令を有している。該ｐｕｓｈ命令やｐｏｐ命令は、レジスタとスタックとのデータのやり取りや、該やり取りに伴うスタックポインタの更新を行う。

【００５０】従って複数のレジスタとスタックでデータ等のやり取りを行う場合には、これらの命令を複数回実行することが必要となる。

【００５１】しかし本発明によれば、前記連続プッシュ命令や前記連続ポップ命令を実行すると、ｐｕｓｈ命令を連続して複数回実行する場合やｐｏｐ命令を連続して複数回実行する場合と同じ効果が得られる。即ち複数のレジスタとスタックとの間のデータのやり取り及び該やり取りに伴うスタックポインタの更新を１命令で実行することが出来る。このため複数のレジスタとスタックとの間のデータの転送を行う場合、ｐｕｓｈ命令あるいはｐｏｐ命令をくり返すことによりオブジェクトコードサイズが増大するのを防ぐことができる。またプログラム実行ステップが長くなることを回避し、無駄なサイクルを消費することなく、割り込み処理及びサブルーチンコール・リターンの処理速度の向上を図ることができる。

【００５２】本発明は、０からｎ－１までのレジスタ番号で特定されたｎ個の汎用レジスタを含み、前記連続プッシュ命令及び連続ポップ命令の少なくとも一方のオブジェクトコードが、前記複数レジスタ特定情報として、前記レジスタ番号のいずれかが指定された最終レジスタ番号を含み、前記実行手段が、レジスタ０から前記最終レジスタ番号で特定されるレジスタまでの複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記スタックポインタ専用レジスタの内容により特定されるメモリアドレスに基づき行うことを特徴とする。

【００５３】通常汎用レジスタが複数ある場合、レジスタを特定するためのアドレスを有している。本発明では０からｎ－１までの連続したレジスタ番号で前記レジスタを特定している。

【００５４】本発明によれば、前記最終レジスタ番号に任意のレジスタ番号を指定することにより、レジスタ０から前記最終レジスタ番号までの連続した複数のレジスタとメモリとの間でデータをプッシュする処理及びポップする処理の少なくとも一方が行われる。従って、レジスタ番号０のレジスタから順番にレジスタを使用するような構造を有するプログラムの実行において、レジスタの待避や復旧を効率的に行うことができる。

【００５５】本発明は、前記実行手段が、前記複数のレジスタのいずれか所与のレジスタの内容を、前記スタックポインタ専用レジスタで特定されるメモリアドレスに基づきメモリに設けられたスタックに書き込む書き込み手段と、前記書き込み手段による前記スタックへの書き込み回数をカウントする書き込み回数カウント手段と、前記カウント手段によってカウントされた前記書き込み回数と前記複数レジスタ特定情報の値を比較する比較手段とを含み、前記書き込み手段が、第一の入力と第二の入力を加算器で加算し書き込み先を特定するための書き込みメモリアドレスを生成する書き込みメモリアドレス生成手段と、前記加算器の第一の入力が、連続専用命令の実行開始時にはスタックポインタ専用レジスタの内容となるように制御し、それ以降は書き込みアドレス生成手段によって生成された書き込みアドレスとなるよう制御する第一の入力制御手段と、前記加算器の第二の入力に前記スタックから１ワードを書き込んだときのオフセット値を出力する第二の入力制御手段と、前記複数レジスタ特定情報から前記書き込み回数を減じた値で特定されるレジスタの内容を、前記書き込みメモリアドレスに

基づき前記スタックに書き込む手段とを含み、前記比較手段の比較結果に基づき、複数のレジスタからの前記スタックへの書き込み及び書き込み終了を制御することを特徴とする。

【００５６】また本発明は、前記命令実行手段が、前記スタックポインタ専用レジスタで特定されるメモリアドレスに基づきメモリに設けられたスタックの内容を読み出し、前記複数のレジスタのいずれか所与のレジスタに格納する読み出し手段と、前記読み出し手段による前記スタックからの読み出し回数をカウントする読み出し回数カウント手段と、前記カウント手段によってカウントされた前記読み出し回数と前記複数レジスタ特定情報の値を比較する比較手段とを含み、前記読み出し手段が、第一の入力と第二の入力を加算器で加算し書き込み先を特定するための書き込みメモリアドレスを生成する書き込みメモリアドレス生成手段と、前記加算器の第一の入力が、連続専用命令の実行開始時にはスタックポインタ専用レジスタの内容となるように制御し、それ以降は読み出しアドレス生成手段によって生成された読み出しアドレスとなるよう制御する第一の入力制御手段と、前記加算器の第二の入力に前記スタックから１ワードを書き込んだときのオフセット値を出力する第二の入力制御手段と、前記読み出しメモリアドレスに基づき前記スタックの内容を読み出し、前記書き込み回数に基づき特定されるレジスタに格納する読み出し手段とを含み、前記比較手段の比較結果に基づき、前記スタックの内容の読み出し及び読み出し終了を制御することを特徴とする。

【００５７】このようにすると連続した値で順序づけて特定される複数のレジスタのスタックへの待避又はスタックから連続した値で特定される複数のレジスタへの復旧をカウント手段と簡単なシーケンス制御のみで実現可能である。従って少ないゲート数の情報処理回路で実現出来るため、ワンチップのマイクロコンピュータ等に適したものである。

【００５８】本発明は、プログラムカウンタ専用のプログラムカウンタレジスタを含み、前記スタックポインタ専用命令群が、サブルーチンへ分岐する命令及び前記サブルーチンからのリターン命令である分岐命令を含み、前記解読手段が、前記分岐命令を解読し、前記命令実行手段が、前記分岐命令を実行する際、メモリに設けられたスタックの所与の第二のエリアへの前記プログラムカウンタレジスタの内容の待避及び前記第二のエリアの内容のプログラムカウンタレジスタへの復帰の少なくとも一方を、前記スタックポインタ専用レジスタにより特定されるメモリアドレスに基づき行う手段と、前記待避及び前記復帰に基づき前記スタックポインタ専用レジスタの内容を更新する手段とを含むことを特徴とする。

【００５９】ここにおいてサブルーチンとは、割り込み処理及び例外処理及びデバッグ処理ルーチン等も含む。従ってサブルーチンへ分岐する命令とは、サブルーチン等をコールする命令、割り込み処理及び例外処理及びデバッグ処理ルーチン等へ分岐するためのソフトウェア割り込み命令、ソフトウェアデバッグ割り込み命令等も含む。またサブルーチンからのリターン命令には、割り込み処理及び例外処理及びデバッグ処理ルーチン等からのリターン命令を含む。

【００６０】通常サブルーチンへ分岐する場合や前記サブルーチンから戻る場合プログラムカウンタの待避や復帰が必要となる。

【００６１】本発明では、前記サブルーチンへ分岐する命令の実行及びサブルーチンからリターンする命令を実行する際に前記プログラムカウンタの待避及び復帰も同時に行う。即ち、本発明の情報処理回路は、サブルーチンへ分岐する命令及び前記サブルーチンからのリターン命令のいずれか一命令でプログラムカウンタの待避及び復帰が出来る回路構成を有している。このためサブルーチンへの分岐やサブルーチンからのリターンに伴って必要となる前記プログラムカウンタの待避及び復帰の命令が不要となり、命令数を削減することができる。また、無駄なサイクルを消費することなく、サブルーチンコール・リターン等の他のルーチンへ分岐時の処理速度の向上を図ることができる。

【００６２】また、ソフトウエア割り込み命令が発生した場合には、例えばＣＰＵ等の情報処理回路の現在の状態を保持するプロセッサーステータスレジスタの待避及び復帰も必要となる。従って、ソフトウエア割り込み命令等の場合は該命令の実行時にプロセッサーステータスレジスタの待避及び復帰も同時に行うようにすることが好ましい。

【００６３】本発明は、連続して順序づけられた複数のレジスタと、いずれかの汎用レジスタに割り当てられたスタックポインタを含む情報処理回路であって、複数レジスタ特定情報をオブジェクトコードに有する連続プッシュ命令及び連続ポップ命令の少なくとも一方の命令のオブジェクトコードを解読して、該オブジェクトコードに基づき制御信号を出力する手段と、前記連続プッシュ命令及び前記連続ポップ命令の少なくとも一方を実行する際、前記複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記制御信号及び前記スタックポインタ専用レジスタの内容により特定されるメモリアドレス及び前記複数レジスタ特定情報とに基づき行う手段を含むことを特徴とする。

【００６４】本発明は、汎用レジスタをスタックポインタとして使用する場合の前記連続プッシュ命令及び前記連続ポップ命令に関する。

【００６５】本発明によれば、前記連続プッシュ命令や前記連続ポップ命令を実行すると、ｐｕｓｈ命令を連続して複数回実行する場合やｐｏｐ命令を連続して複数回

実行する場合と同じ効果が得られる。即ち複数のレジスタとスタックとの間のデータのやり取り及び該やり取りに伴うスタックポインタの更新を１命令で実行することが出来る。このため複数のレジスタとスタックとの間のデータの転送を行う場合、ｐｕｓｈ命令あるいはｐｏｐ命令をくり返すことによりオブジェクトコードサイズが増大するのを防ぐことができる。またプログラム実行ステップが長くなることを回避し、無駄なサイクルを消費することなく、割り込み処理及びサブルーチンコール・リターンの処理速度の向上を図ることができる。

【００６６】本発明はの情報処理回路はＲＩＳＣ方式であることを特徴とする。

【００６７】ＲＩＳＣ方式の情報処理回路は、ハードウエアを小型化して高速化を図ることを目的として設計されている。このため汎用レジスタを多く有しており、命令セットを汎用性の高いものに絞ることで命令数の削減を図っている。

【００６８】従ってＲＩＳＣ方式の情報処理回路では、スタックポインタを汎用レジスタに割り当て、スタックポインタを扱う場合は汎用レジスタを扱う命令セットを用いて処理を行っていた。しかしこの様な方法では命令長が大きくなりメモリの使用効率がよくない。

【００６９】本発明によれば、ＲＩＳＣ方式の情報処理回路において、命令長を削減することが出来、メモリの使用効率を上げることが出来る。

【００７０】本発明の情報処理回路は、固定長の命令を解読し、該命令に基づき実行処理を行うことを特徴とする。

【００７１】固定長命令を用いると可変長命令を用いる場合に比べ、命令のデコードに要する時間を短縮でき、情報処理回路の回路規模を小さくすることが出来る。固定長命令を採用する場合、命令に冗長な部分が出来るのを防ぎメモリを効率よく使用するためには、各命令に必要なビット数はばらつきが少なく、出来るだけ短いほうが好ましい。

【００７２】本発明によれば、一般に命令長が長くなりがちなスタックポインタを取り扱う命令の命令長を短くすることができる。従って固定長命令を採用した場合であっても命令に冗長な部分が出来るのを防ぎ、メモリを効率よく使用することが出来る。

【００７３】本発明のマイクロコンピュータは、前述した本発明の情報処理回路と記憶手段と外部との入出力を行う手段とを含むことを特徴とする。

【００７４】本発明によれば、処理速度が速くメモリの使用効率のよい、マイクロコンピュータを提供することが出来る。

【００７５】本発明のマイクロコンピュータは、前記スタックポインタに関連づけてオート変数の記憶領域が確保される構造を有する言語のプログラムが実行されることを特徴とする。

【００７６】スタックポインタに関連づけてオート変数の記憶領域が確保される構造を有するする言語として、例えばＣ言語がある。この様な言語のプログラムの処理を本発明のマイクロコンピュータが行う場合、処理速度及びメモリの使用効率を効果的に向上させることが出来る。

【００７７】本発明の電子機器は、前述した本発明のマイクロコンピュータを含むことを特徴とする。

【００７８】本発明によれば、処理速度が速くメモリの使用効率のよい情報処理回路を内蔵しているため、安価で高機能な電子機器を提供することが出来る。

【００７９】

【発明の実施の形態】以下、本実施例を図面に基づき説明する。

【００８０】（実施例１）

（１）本実施例のＣＰＵの構成

本実施例のＣＰＵはパイプラインとロード・ストア型のアーキテクチュアによって、ほとんど全ての命令を１サイクルで実行する。全ての命令は１６ビットの固定長で記述されており、本実施例のＣＰＵの処理する命令は極めて小さいオブジェクトコードサイズを実現している。

【００８１】特に本実施例のＣＰＵは、スタックポインタを取り扱う処理を効率よく記述し実行するためにスタックポインタ専用のレジスタを有し、該スタックポインタ専用レジスタを暗黙のオペランドとするオブジェクトコードを有するスタックポインタ専用命令群の命令セットを解読、実行出来るよう構成されている。

【００８２】図１は、本実施例のＣＰＵの回路構成の概略を説明するための図である。

【００８３】本ＣＰＵ１０は、汎用レジスタ１１、プログラムカウンタが格納されているＰＣ１２、プロセッサーステータスレジスタ（ＰＳＲ）１３、スタックポインタ専用のレジスタであるＳＰ１４を含むレジスタセットと、命令デコーダ２０、即値生成器２２、アドレス加算器３０、ＡＬＵ４０、ＰＣインクリメンタ４４及び各種内部バス７２、７４、７６、７８、各種内部信号線８２、８４、８６、８８等を含む。

【００８４】前記命令デコーダ２０は、入力したオブジェクトコードを解読し、該命令を実行するために必要な処理を行い、必要な制御信号を出力する。なお、該命令デコーダ２０は前記スタックポインタ専用命令のオブジェクトコードを解読して該命令に基づき制御信号を出力する前記解読手段としても機能する。

【００８５】即値生成器２２は、オブジェクトコードに含まれた即値に基づき、実行時に使用する３２ビットの即値データを生成したり、各命令の実行に必要な０、±１、±２、±４のconstantデータを生成したりする。ＰＣインクリメンタ４４は、命令の実行サイクルに基づきＰＣ１２に格納されたプログラムカウンタの更新を行う。アドレス加算器３０は、各種レジスタに格納されて

いる情報や即値生成器２２で生成される即値データの加算を行いメモリからデータを読み出す際に必要なアドレスデータを生成する。ＡＬＵ４０は数値演算や論理演算を行う。

【００８６】また、該ＣＰＵは内部に各種バスや信号線を含んでいる。ＰＡ＿ＢＵＳ７２やＰＢ＿ＢＵＳ７４はＡＬＵ４０の入力信号を伝送する機能等を有する。ＷＷ＿ＢＵＳ７６はＡＬＵ４０の演算結果を取り出して汎用レジスタに伝送する機能等を有する。ＸＡ＿ＢＵＳ７８は汎用レジスタ１１やＳＰ１４から取り出したアドレスデータを伝送する機能等を有する。ＩＡ信号線８２は、ＣＰＵ内部の各部から外部のＩ＿ＡＤＤＲ＿ＢＵＳ９２へアドレスデータを伝送する。ＤＡ信号線８４は、ＣＰＵ内部の各部から外部のＤ＿ＡＤＤＲ＿ＢＵＳ９６へアドレスデータを伝送する。ＤＩＮ信号線８６は、ＣＰＵ外部のＤ＿ＤＡＴＡ＿ＢＵＳ９８からＣＰＵ内部の各部へデータを伝送する。ＤＯＵＴ信号線８８は、ＣＰＵ内部の各部から外部のＤ＿ＤＡＴＡ＿ＢＵＳ９８へデータを伝送する。ＩＡ入力切換８３はＩＡ信号線８２へ出力される各種信号（ＰＡ＿ＢＵＳ７２、ＷＷ＿ＢＵＳ７６、ＰＣ１２、ＰＣ＋２）の切換を行う。ＤＯＵＴ入力切換８９はＤＯＵＴ信号線８８へ出力される各種信号（ＰＡ＿ＢＵＳ７２、ＷＷ＿ＢＵＳ７６、ＰＣ１２、ＰＣ＋２）の切換を行う。

【００８７】また、ＣＰＵ１０の前記各部は前記命令デコーダ２０の出力する制御信号に基づき、命令の実行をおこなうもので、スタックポインタ専用命令群を、前記制御信号及び前記スタックポインタ専用レジスタの内容に基づき実行する前記実行手段としても機能する。

【００８８】本ＣＰＵ１０は、１６ビットの命令データバス（Ｉ＿ＤＡＴＡ＿ＢＵＳ）９４、命令データアクセスのための命令アドレスバス（Ｉ＿ＡＤＤＲ＿ＢＵＳ）９２と、３２ビットのデータバス（Ｄ＿ＤＡＴＡ＿ＢＵＳ）９８と、データアクセスのためのデータアドレスバス（Ｄ＿ＡＤＤＲ＿ＢＵＳ）９６と、コントロール信号のための図示しないコントロールバスを介して外部と信号のやり取りを行う。

【００８９】（２）本実施例のＣＰＵ有するレジスタセットの説明

次に本実施例のＣＰＵが有するレジスタセットの概要について、必要な部分について説明する。

【００９０】図２に本実施例のＣＰＵの持つレジスタセットを示す。本実施例のＣＰＵは、汎用レジスタ１１を１６本と、ＰＣ１２、ＰＳＲ１３、ＳＰ１４、図示しないＡＬＲ（算術演算ローレジスタ）１５、図示しないＡＨＲ（算術演算ハイレジスタ）１６を含むレジスタセットを有している。

【００９１】前記汎用レジスタ１１は機能的に等価な３２ビットのレジスタであり、Ｒ０からＲ１５と名付けらている。該汎用レジスタ１１はデータ演算時及びアドレス計算時に使用される。

【００９２】またＰＣ１２は、３２ビット長のインクリメンタルカウンタであり、現在実行中の命令のアドレスであるプログラムカウンタを保持している。本文中で、レジスタ名を指すときはＰＣといい、ＰＣに格納された値をさすときはプログラムカウンタという。

【００９３】該ＰＣ１２は、ロード命令等で直接アクセスすることは出来ない。ｃａｌｌ命令、ｉｎｔ命令や割り込み、例外が発生すると、プログラムカウンタはＰＣ１２から読み出されスタックに退避される。この様に分岐命令が実行されると飛びさきアドレスがＰＣに設定される。条件分岐命令で分岐する場合も同様である。そして、ｒｅｔ命令やｒｅｔｉ命令により、戻りさき命令アドレスがスタックより読み出され、ＰＣ１２に復帰される。

【００９４】ＰＳＲ（プロセッサステータスレジスタ）１３は、フラグが割り当てられている３２ビットレジスタであり、ＣＰＵの現在の状態を保持している。ｉｎｔ命令、割り込み、例外等が発生すると、それぞれの処理ルーチンに分岐する際に、その時のＰＳＲの状態がスタックに退避される。逆にｒｅｔｉ命令の実行で、退避されていた値がＰＳＲに復帰される。

【００９５】ＳＰ１４は３２ビットのスタックポインタ専用のレジスタで、スタックの先頭番地をさすスタックポインタが格納されている。本文中で、レジスタ名を指すときはＳＰといい、ＳＰに格納された値をさすときはスタックポインタという。但し、スタックポインタは常にワードの境界を指しているため、前記スタックポインタの下位の２ビットは常に０である。

【００９６】該スタックポインタは、本実施例で用意しているスタックポインタ専用命令群に含まれる各種命令の実行や、トラップの発生に伴い更新される。スタックポインタの更新を行うスタックポインタ専用命令としては、ｃａｌｌ命令やｒｅｔ命令等の他のルーチンへ分岐する命令、スタックポインタ移動命令、ｐｕｓｈｎ命令、ｐｏｐｎ命令等がある。例えばｃａｌｌ命令が実行されると、まずスタックポインタがワードサイズ（４）だけデクリメント（－４）され、ＰＣ１２がスタックに退避される。また、ｒｅｔ命令が実行されると、逆にスタックより戻りさきアドレスがＰＣにロードされ、スタックポインタはワードサイズ分インクリメント（＋４）される。ｉｎｔ命令が実行されたり、または、割り込み、例外等が発生したときには，スタックにＰＣやＰＳＲの値が以下の手順で退避される。

【００９７】１）ＳＰ＝ＳＰ－４

２）スタックポインタで指されるスタックの先頭番地にＰＣを退避する。

【００９８】３）ＳＰ＝ＳＰ－４

４）スタックポインタで指されるスタックの先頭番地にＰＳＲを退避する。

【００９９】ｒｅｔｉ命令が実行されると、上記とは逆の処理を行ってＣＰＵは以前の状態に戻る。このようにｃａｌｌ命令やｒｅｔ命令やｉｎｔ命令が実行されると、該実行に基づきスタックポインタが更新される。各命令の詳細については後述する。

【０１００】また、本実施例においてトラップとは、命令の実行に非同期に発生する割り込みと命令の実行により発生する例外を合わせたものをいう。トラップが発生すると、ＣＰＵはプログラムカウンタ（ＰＣ）とプロセッサステータスレジスタ（ＰＳＲ）をスタックに退避してからトラップテーブルよりベクタテーブルを読み出し、そのトラップに対応する処理ルーチンへ分岐する。トラップの発生に伴って，ＰＳＲ中のＩＥビット（割り込みイネーブル）はクリアされ、それ以上のマスク可能割り込みの発生が禁止される。再び、マスク可能な外部割り込みをイネーブルにするには，ＰＳＲに対するロード命令を使ってＰＳＲのＩＥビットに１を書き込んで設定する。

【０１０１】トラップ処理から元のルーチンに復帰するには、ｒｅｔｉ命令を使う。ｒｅｔｉ命令を実行すると、ＣＰＵはスタックからＰＳＲ、ＰＣの順に読み出して、ＰＳＲを元の値に戻すとともに、戻りアドレスに分岐する。なお、例外にはデバッグ例外、アドレス不整列例外、オーバーフロー例外、ゼロ除算例外等がある。

【０１０２】ＡＬＲ（算術演算ローレジスタ）１５及びＡＨＲ（算術演算ハイレジスタ）１６については説明を省略する。

【０１０３】ＣＰＵの持つ特殊レジスタＰＳＲ１３、ＳＰ１４、ＡＬＲ１５、ＡＨＲ１６はロード命令を使って汎用レジスタとの間でデータ転送を行うことが出来る。各レジスタは、特殊レジスタ番号を持っており、この番号を使ってアクセスされる。

【０１０４】

特殊レジスタ名	特殊レジスタ番号	アセンブラの記述方法
プロセッサステータスレジスタ	0	%ＰＳＲ
ＳＰ	1	%ＳＰ
算術演算ローレジスタ	2	%ＡＬＲ
算術演算ハイレジスタ	3	%ＡＨＲ

（３）スタック及びスタックポインタについての説明

スタックはメモリにもうけられた一時記憶領域で、連続した領域の下方からデータが棚状に積み重ねて記憶される。スタックポインタは、スタックメモリの一番上つまり最後に記憶されたデータのアドレスを示している。

【０１０５】スタックポインタの一般的な動作について図３を用いて説明する。

【０１０６】図３の１００はメモリに設けられたスタック領域を表している。斜線部分１０２が最後に格納されたデータであるとすると、ＳＰ１４に格納されたスタックポインタは、該データのメモリアドレス１０００を示している。なお、斜線部分１０２の下方領域１０４は既にデータが格納されている領域で、斜線部分１０２の上方領域１０６はこれからデータが格納される領域を示している。

【０１０７】スタックポインタは常にワードの境界を指しているため、スタックに情報を書き込む時はＳＰ１４の格納されたスタックポインタを４だけ上に動かして、該スタックポインタのさす場所に情報を格納する。またスタックに格納されている情報を取り出すときは、現在ＳＰ１４が示すアドレスの情報をとりだして、ＳＰ１４に格納されたスタックポインタを４だけ下に動かす。このようにスタックポインタは常に最後にスタックに格納された情報の格納アドレスを示している。

【０１０８】（４）スタックポインタ専用命令群についての説明

通常ＲＩＳＣ方式のＣＰＵでは汎用レジスタをスタックポインタとして使用するが、本実施例ではスタックポインタ専用のレジスタであるＳＰ１４を有しており、前記スタックポインタ専用命令群の操作対象となる。

【０１０９】前記スタックポインタ専用命令群は、スタックポインタ専用のレジスタであるＳＰ１４を暗黙のオペランドとし、ＳＰ１４をもちいて操作をおこなう複数の命令の総称である。該スタックポインタ専用命令群は、ＳＰ相対ロード命令（ｌｄ等）、スタックポインタ移動命令（ａｄｄ、ｓｕｂ）、サブルーチンへ分岐する命令（ｃａｌｌ等）及びリターン命令（ｒｅｔ等）等の他のルーチンへ分岐する命令、連続プッシュ命令（ｐｕｓｈｎ）、連続ポップ命令（ｐｏｐｎ）を含む。

【０１１０】これらの命令に共通するのは、スタックポインタ専用命令であるため、オブジェクトコードにスタックポインタを特定するための情報が必要ないということである。従ってスタックポインタを用いる処理を短い命令長で効率よく記述することができるという効果も共通する。

【０１１１】また、これらの命令を用いるとメモリに設けられたスタックに記憶されている情報の処理を効率よく行うことができる。また、割り込み処理及びサブルーチンコール・リターンの処理を効率よく行うことが出来る。

【０１１２】ここで、サブルーチンがコールされる場合の前記スタックポインタ専用命令の使用例とメモリにもうけられたスタックの使用状態及びスタックポインタの状態について図４を用いて説明する。図４のメインプログラム５００とサブルーチン５２０は、Ｃコンパイラが作成したオブジェクトコードで記述されたプログラムで

ある。５４０はメモリ上のスタックの状態を示している。５０２は汎用レジスタＲ０～Ｒ３を使用する処理が行われていることを示している。５０６はサブルーチンコール命令を示している。メインプログラム５００でサブルーチンコール命令が実行されるサブルーチンコール命令前、即ち５０４に示す前の命令が実行された後スタックポインタ（ＳＰ）はメモリ上のスタック５４０上の①のアドレスを指していたとする。前記サブルーチンコール命令が実行されると、実行の制御はサブルーチン５２０にわたる。このとき本実施例では、前記スタックポインタ専用命令であるサブルーチンへ分岐する命令（ｃａｌｌ命令）が実行される。該命令が実行されると、スタックポインタ（ＳＰ）の値は自動的に－４だけインクリメントされ（図４の②ＳＰ参照）、該スタックポインタ②ＳＰが指しているスタック上のエリア５４４にメインプログラムへのリターンアドレスが格納される。

【０１１３】そして、サブルーチン５２０側では、実行の最初にメインプログラムで使用されていた汎用レジスタＲ０～Ｒ３に格納されていた値をスタックに転送する処理を行う。５２４は、汎用レジスタＲ０～Ｒ３に格納されていた値をスタックに待避する処理を行うスタックポインタ専用命令である連続プッシュ命令（ｐｕｓｈｎ）を示している。該命令が実行されると汎用レジスタＲ０～Ｒ３に格納されていた値が連続的にスタックに転送され、図４の５５０に示すように、スタック５４０に格納される。この処理の実行が終わる（５２４③）と、スタックポインタ（ＳＰ）は５４６を指している（③ＳＰ）。

【０１１４】次にサブルーチン５２０では、サブルーチンで使用するオート変数領域の確保を行う。５２６に示したａｄｄ命令は、スタックポインタ専用命令であるスタックポインタ移動命令であり、スタックポインタを上方に移動させサブルーチン５２０で使用するスタック領域を確保する。該命令が実行されるとスタックポインタ（ＳＰ）はＸバイトだけ上方の位置５４８に移動し（④ＳＰ）、注２）に示すサブルーチンが使用するオート変数の領域が確保される。

【０１１５】５２８はサブルーチン５２０でオート変数と汎用レジスタＲ０～Ｒ３を用いた処理を行うことを示している。このときスタックポインタの位置は⑤ＳＰを指しており、オート変数のロードは、スタックポインタ専用命令であるＳＰ専用ロード命令をもちいて行われる。

【０１１６】５２９は、メモリ上に格納されたオート変数Ｓ１を汎用レジスタＲ１に転送する前記ＳＰロード命令を示している。前記オート変数Ｓ１はスタックポインタ（⑤ＳＰ）からＹバイトだけオフセット値を有する場所に格納されている。前述したようにサブルーチンの５２８の処理の最中にはスタックポインタは移動しないので、オート変数のメモリアドレスは、スタックポインタ＋オフセット値で特定され、前記ＳＰ専用ロード命令を用いて効率良く汎用レジスタとのやり取りを行うことができる。

【０１１７】また、サブルーチン５２０からメインプログラム５００に戻る前には、スタックに待避されていた汎用レジスタＲ０～Ｒ３の値を汎用レジスタＲ０～Ｒ３に復帰させ、スタックポインタがメインルーチンへのリターンアドレスが格納されているエリア５４４を指すようにしなければならない。そのためにまず、５２６のスタックポインタ移動命令で移動させたスタックポインタをもとの位置に戻してやる。５３０に示したｓｕｂ命令は、スタックポインタ専用命令であるスタックポインタ移動命令であり、スタックポインタを下方に移動させる。該命令が実行されると、スタックポインタは５４６に移動する（⑥ＳＰ参照）。次にスタックに格納されている情報５５０を汎用レジスタＲ０～Ｒ３に復旧する処理を行う。５３２は、スタックの情報を汎用レジスタＲ０～Ｒ３に転送するする処理を行うスタックポインタ専用命令である連続ポップ命令（ｐｏｐｎ）を示している。該命令が実行されるとスタックに格納されていた値５５０が汎用レジスタＲ０～Ｒ３に連続的に転送され、図４の５５０に示すように、スタック５４０に格納される。この処理の実行が終わる（５３２⑦）と、スタックポインタは５４４を指している（⑦ＳＰ参照）。

【０１１８】５３４はリターン命令を示している。前記リターン命令が実行されると、実行の制御はメインプログラム５００にわたる。このとき本実施例では、前記スタックポインタ専用命令であるリターン命令（ｒｅｔ命令）が実行される。該命令が実行されると、⑦ＳＰが指すスタックエリアに格納されたメインプログラムへのリターンアドレスが示す命令に分岐する、即ちメインプログラム５００の次の命令５０７に戻る。そしてスタックポインタの値は自動的に＋４だけインクリメントされメインプログラム５００が使用するスタックの先頭エリアに移動する（図４の⑧ＳＰ参照）。

【０１１９】以下前記各スタックポインタ専用命令についてそれぞれ命令の説明及び該命令を実行するための回路構成及び実行時の動き等について詳細に説明する。

【０１２０】（５）スタックポインタ相対ロード命令

前述したようにＣコンパイラはオート変数の領域をスタックポインタに関連づけて記憶するオブジェクトコードを作成する。具体的には、Ｃコンパイラはスタックポインタからオフセットが Ｘのところに Ｙバイトだけオート変数の領域を確保する仕様になっている。

【０１２１】図５は、メモリ上のオート変数をレジスタに転送する処理を説明するための図である。オート変数ａはワードデータ、オート変数ｂ、ｃはハーフワードデータ、オート変数ｄ～ｇはバイトデータである。ＳＰに格納されたスタックポインタはスタック上のオート変数が格納される領域６００の先頭領域のアドレスである１

０００を示している。オート変数ａを格納するためのエリアとして前記スタックポインタをメモリアドレスとするスタック上のエリアが、オート変数ｂ、ｃを格納するためのエリアとしてそれぞれスタックポインタ＋２、スタックポインタ＋４をメモリアドレスとするスタック上のエリアが、オート変数ｄ～ｇを格納するためのエリアとしてそれぞれスタックポインタ＋５、スタックポインタ＋６、スタックポインタ＋７、スタックポインタ＋８をメモリアドレスとするスタック上のエリアが確保されている。そして所与の処理を実行する際には、スタックポインタ＋オフセット値のメモリアドレスで特定されるオート変数と汎用レジスタ間でデータの転送処理が必要となる。

【０１２２】本実施例のＣＰＵでは、このような転送処理を短いオブジェクトコードで記述し効率良く実行するために、スタックポインタ専用命令の一つであるＳＰ相対ロード命令として、次に示す命令セットを用意している。

【０１２３】

ｌｄ．ｂ　％Ｒｄ、［％ｓｐ＋ｉｍｍ６］　…（１）
ｌｄ．ｕｂ　％Ｒｄ、［％ｓｐ＋ｉｍｍ６］…（２）
ｌｄ．ｈ　％Ｒｄ、［％ｓｐ＋ｉｍｍ７］　…（３）
ｌｄ．ｕｈ　％Ｒｄ、［％ｓｐ＋ｉｍｍ７］…（４）
ｌｄ．ｗ　％Ｒｄ、［％ｓｐ＋ｉｍｍ８］　…（５）
ｌｄ．ｂ　［％ｓｐ＋ｉｍｍ６］、％Ｒｓ　…（６）
ｌｄ．ｈ　［％ｓｐ＋ｉｍｍ７］、％Ｒｓ　…（７）
ｌｄ．ｗ　［％ｓｐ＋ｉｍｍ８］、％Ｒｓ　…（８）

（１）～（８）は命令コードをアセンブラで記述したものである。（１）はスタックからバイトデータをサイン拡張してレジスタに転送する命令であり、（２）はスタックからバイトデータをゼロ拡張してレジスタに転送する命令であり、（３）はスタックからハーフワードデータをサイン拡張してレジスタに転送する命令であり、（４）はスタックからハーフワードデータをゼロ拡張してレジスタに転送する命令であり、（５）はスタックからワードデータをレジスタに転送する命令であり、（６）はレジスタからバイトデータをスタックに転送する命令であり、（７）はレジスタからハーフワードデータをスタックに転送する命令であり、（８）はレジスタからワードデータをスタックに転送する命令である。

【０１２４】［％ｓｐ＋ｉｍｍ６］、［％ｓｐ＋ｉｍｍ７］、［％ｓｐ＋ｉｍｍ８］は、それぞれ即値オフセット情報を表しており、該即値オフセット情報に基づき実行時に生成されるオフセット値とＳＰ１４に格納されたスタックポインタの値を加算してメモリアドレスが生成される。［％ｓｐ＋ｉｍｍ６］はメモリ上のデータのデータサイズがバイトの時、［％ｓｐ＋ｉｍｍ７］はメモリ上のデータのデータサイズがハーフワードの時、［％ｓｐ＋ｉｍｍ８］はメモリ上のデータのデータサイズがワードの時の即値オフセット情報である。これらはいずれも、後述する図６（Ａ）に示す用にオブジェクトコード上では６ビットの即値オフセット情報６１４として記述されている。しかし命令実行時には、ｉｍｍ７の場合（即ちデータサイズがハーフワードの場合）、即値オフセット情報６１４は１ビット左論理シフトしてスタックポインタに加算すべきオフセット値が生成される。また、ｉｍｍ８の場合（即ちデータサイズがワードの場合）、即値オフセット情報６１４は２ビット左論理シフトしてスタックポインタに加算すべきオフセット値が生成される。

【０１２５】ここにおいてスタックとはメモリに設けられた一時記憶領域をいい、前記メモリアドレスで、例えば図５に示すスタック上のオート変数（ａ～ｇ）の位置を指定することができる。

【０１２６】図６（Ａ）は、前記（１）～（８）のＳＰ相対ロード命令のビットフィールド６１０である。図６（Ａ）に示すようにＳＰ相対ロード命令は、操作機能がメモリとレジスタ間のデータの転送であることを示すオペコード６１２（６ビット）、即値で指定された即値オフセット情報（６ビット）６１４と、転送対象となる汎用レジスタのレジスタ番号６１６（４ビット）を含み、１６ビットのオブジェクトコードを有している。前記オペコード６１２はＳＰ１４を操作対象とするロード命令であることを示す共通のコードと前記データサイズ、サイン拡張及びゼロ拡張の別に応じて与えられる異なるコードとを含んでいる。従ってオペコードによりＳＰ１４に格納されたスタックポインタを操作対象とすることがわかるため、オブジェクトコードのオペランドにスタックポインタに関する情報を必要としない。即値オフセット情報６１４は転送対象となるデータのスタックポインタからのオフセット値を生成するための情報である。これは、前述したようにデータサイズを問わず６ビットで指定される。転送対象となるレジスタのアドレス６１６には、前記（１）～（５）の場合はスタックからよみだされたデータが格納されるレジスタのレジスタ番号が、前記（６）～（８）の場合はスタックに書き込まれるデータが格納されているレジスタのレジスタ番号が入っている。

【０１２７】図６（Ｂ）はスタックポインタとして汎用レジスタを用いた場合に使用する汎用レジスタを操作対象とするロード命令（以下汎用ロード命令という）のオブジェクトコードのビットフィールド６２０の例を示している。

【０１２８】図６（Ｂ）に示すように汎用ロード命令は、操作機能がメモリと汎用レジスタ間のデータの転送であることを示すオペコード６２２（６ビット）、即値で指定された即値オフセット情報６２４（６ビット）とスタックポインタとして使用する汎用レジスタを特定する第一のレジスタ番号６２６（４ビット）と、転送対象となる汎用レジスタを特定する第二のレジスタ番号６２８（４ビット）を含み、２０ビットのオブジェクトコー

ドを有している。汎用マイクロコンピュータでは命令長は８ビット単位なので２４ビット若しくは３２ビット命令になる。

【０１２９】図６（Ａ）（Ｂ）は、いずれもスタックポインタにオフセット値を加算して特定されるメモリアドレスとレジスタの間でのデータの転送処理を行う場合に使用する命令のオブジェクトコードを示しているが、同図に示すように、ＳＰ相対ロード命令は汎用ロード命令に比べて短いオブジェクトコードで記述することができる。

【０１３０】以下、スタックからレジスタにデータを転送する命令として（５）の命令（スタックからレジスタにワードデータを読み出す命令なので、以下ワードデータ読み出しのＳＰ相対ロード命令という）、レジスタからスタックにデータを転送する命令として（８）の命令（レジスタからスタックにワードデータを取り込む命令なので、以下ワードデータ書き込みのＳＰ相対ロード命令という）を例にとりこれらの命令を実行するための構成及び実行時の動作について説明する。

【０１３１】まず図１を用いてこれらの命令を実行するために必要なハードウエア構成について説明する。これらの命令は外部のメモリ（ＲＯＭ）５２よりＩ＿ＤＡＴＡ＿ＢＵＳ９４を介して伝送され、ＣＰＵ１０の命令デコーダ２０に入力される。該命令デコーダ２０で、命令が解読され命令の実行に必要な図示しない各種信号が出力される。また前記即値生成器２２は、データサイズに応じて前記即値オフセット情報６１４の左論理シフトを行い、必要に応じてサイン拡張及びゼロ拡張行い実行に使用するオフセット値を生成し、ＰＢ＿ＢＵＳ７４に出力する。ＳＰ１４はスタックポインタを格納しており、この値はアドレス加算器３０の入力に接続されたＸＡ＿ＢＵＳ７８に出力できる。アドレス加算器３０のもう一方の入力は、即値生成器２２の出力であるＰＢ＿ＢＵＳ７４に接続されている。アドレス加算器３０の出力（ＡＤＤＲ）は、ＤＡ信号線８４を介して外部のＤ＿ＡＤＤＲ＿ＢＵＳ９６に接続されている。

【０１３２】バスコントロールユニット（ＢＣＵ）６０は、ＣＰＵから出力される各種リクエスト信号（外部バスに出力された信号等）に基づき、スタックエリアを含むメモリ（ＲＡＭ、ＲＯＭ）５０、５２とのデータの入出力を制御し、ＲＥＡＤ、ＷＲＩＴＥ制御信号を出力する。

【０１３３】まずワードデータ読み出しのＳＰ相対ロード命令の実行時の動作について説明する。

【０１３４】ワードデータ読み出しのＳＰ相対ロード命令が実行されると、ＳＰ１４に格納されたスタックポインタの値と、前記即値オフセット情報６１４に基づき即値生成器２２が生成したオフセット値が加算され、メモリ読み出し用のメモリアドレスが生成される。そして該メモリアドレスに基づきメモリ上の情報が読み出され、オブジェクトコードの前記レジスタ番号６１６で特定される汎用レジスタに転送される。

【０１３５】図７はワードデータ読み出しのＳＰ相対ロード命令の動作を説明するためのフローチャート図である。

【０１３６】前記命令の実行の最初にＳＰ１４に格納されているスタックポインタがＸＡ＿ＢＵＳ７８に出力される（ステップ２１０）。また、即値生成器２２が即値オフセット情報から生成したオフセット値immがＰＢ＿ＢＵＳ７４に出力される（ステップ２１２）。アドレス加算器３０は、前記ＸＡ＿ＢＵＳ７８上の値と前記ＰＢ＿ＢＵＳ７４上の値を加算し、結果（ＡＤＤＲ）であるメモリの読み出しアドレスをＤＡ信号線８４を介してＤ＿ＡＤＤＲ＿ＢＵＳ９６に出力する（ステップ２１４、ステップ２１６）。そしてＣＰＵからＢＣＵ６０へのデータ読み出しリクエスト信号がアクティブになり、外部メモリのリードサイクルを実行する（ステップ２１８）。即ち前記ＢＣＵ６０は該リクエスト信号に基づき、前記読み出しアドレスをメモリアドレスとしてメモリからデータが読み出され、Ｄ＿ＤＡＴＡ＿ＢＵＳ９８に出力されるよう制御する。Ｄ＿ＤＡＴＡ＿ＢＵＳ９８上のデータはＤＩＮ信号線８６を介してＷＷ＿ＢＵＳ７６に出力される（ステップ２２０）。そしてＷＷ＿ＢＵＳ７６上の値は命令コードの転送対象となるレジスタ（Ｒｓ／Ｒｄ）のアドレス（４ビット）６１６で指定されたレジスタ番号を有するレジスタ（％Ｒｄ）に格納される（ステップ２２２）。

【０１３７】次にワードデータ書き込みのＳＰ相対ロード命令の実行時の動作について説明する。

【０１３８】ワードデータ書き込みのＳＰ相対ロード命令が実行されると、ＳＰ１４に格納されたスタックポインタの値と、前記即値オフセット情報６１４に基づき即値生成器２２が生成したオフセット値が加算され、メモリ書き込み用のメモリアドレスが生成される。そしてオブジェクトコードの前記レジスタ番号６１６で特定される汎用レジスタに格納されている情報が、該メモリアドレスに基づき特定されるメモリ上のエリアに転送される。

【０１３９】図８はワードデータ書き込みのＳＰ相対ロード命令の動作を説明するためのフローチャート図である。

【０１４０】前記命令の実行の最初にＳＰ１４に格納されているスタックポインタがＸＡ＿ＢＵＳ７８に出力される（ステップ２３０）。また、即値生成器２２が即値オフセット情報から生成したオフセット値ｉmmがＰＢ＿ＢＵＳ７４に出力される（ステップ２３２）。アドレス加算器３０は、前記ＸＡ＿ＢＵＳ７８上の値と前記ＰＢ＿ＢＵＳ７４上の値を加算し、結果（ＡＤＤＲ）であるメモリへの書き込みアドレスをＤＡ信号線８４を介してＤ＿ＡＤＤＲ＿ＢＵＳ９６に出力する（ステップ２３

４、ステップ２３６）。また、命令コードの転送対象となるレジスタ（Ｒｓ／Ｒｄ）のアドレス（４ビット）６１６で指定されたレジスタ番号を有するレジスタ（％Ｒｄ）に格納されているデータがＰＡ＿ＢＵＳ７２に出力される（ステップ２３８）。ＰＡ＿ＢＵＳ７２上のデータはＤＯＵＴ信号線８８を介して、Ｄ＿ＤＡＴＡ＿ＢＵＳ９８に出力される（ステップ２４０）。そしてＣＰＵからＢＣＵ６０へのデータ書き込みリクエスト信号がアクティブとなり、外部メモリのライトサイクルを実行する（ステップ２４２）。即ち前記ＢＣＵ６０は前記リクエスト信号に基づき、前記書き込みアドレスをメモリアドレスとして、Ｄ＿ＤＡＴＡ＿ＢＵＳ９８で転送されてきたデータをメモリ５０に書き込む動作を制御する。

【０１４１】（６）スタックポインタ移動命令

図９（Ａ）～（Ｆ）は、複数のルーチンにわたってプログラムが実行される場合の各ルーチンによるメモリ上のスタックの使用状態及びスタックポインタの状態を説明するための図である。

【０１４２】図９（Ａ）に示すＭＡＩＮルーチン２１０の所与の処理ａの実行時におけるメモリ上のスタック領域の状態及びスタックポインタ（ＳＰ１４に格納されている値）の状態を図９（Ｄ）に示している。２２２はＭＡＩＮルーチン２１０で使用するためのスタック領域を示しており、スタックポインタは２２２の先頭アドレス２３２を示している。

【０１４３】図９（Ｂ）のＳＵＢ１ルーチン２１２は、前記ＭＡＩＮルーチン２１０から呼ばれて実行されるサブルーチンである。該ＳＵＢ１ルーチン２１２の所与の処理ｂ２１３の実行時におけるメモリ上のスタック領域の状態及びスタックポインタ（ＳＰに格納されている値）の状態を図９（Ｅ）に示している。２２４はＳＵＢ１ルーチン２１２で使用するためのスタック領域を示しており、スタックポインタは２２４の先頭アドレス２３４を示している。

【０１４４】図９（Ｃ）のＳＵＢ２ルーチン２１４は、前記ＳＵＢ１ルーチン２１０から呼ばれて実行されるサブルーチンである。該ＳＵＢ２ルーチン２１４の所与の処理ｃ２１５の実行時におけるメモリ上のスタック領域の状態及びスタックポインタ（ＳＰに格納されている値）の状態を図９（Ｆ）に示している。２２６はＳＵＢ２ルーチン２１４で使用するためのスタック領域を示しており、スタックポインタは２２６の先頭アドレス２３６を示している。

【０１４５】このように複数のサブルーチンにわたって実行が行われる場合、各ルーチンで使用するスタック領域が移動するため、それに伴いスタックポインタの値を、各ルーチンで使用するスタック領域の先頭に移動することが行われる。

【０１４６】本実施例のＣＰＵでは、このようなスタックポインタの移動処理を短いオブジェクトコードで記述し効率良く実行するために、スタックポインタ専用命令の一つであるスタックポインタ移動命令として、次に示す命令セットを用意している。

【０１４７】

```
add　％ｓｐ、ｉｍｍ１２　　…（９）
ｓｕｂ　％ｓｐ、ｉｍｍ１２　　…（１０）
```

（９）（１０）は命令コードをアセンブラで記述したものである。（９）はＳＰ１４に格納されたスタックポインタに対する即値加算命令であり、（１０）はＳＰ１４に格納されたスタックポインタに対する即値減算命令である。ｉｍｍ１２は、命令のオブジェクトコードに含まれた１０ビットの即値を２ビット左方向にシフトした後、ゼロ拡張されて３２ビットデータとなり、ＳＰ１４に格納されたスタックポインタとの演算に用いられる。

【０１４８】図１０（Ａ）は、前記（９）（１０）のスタックポインタ移動命令のビットフィールド６３０である。図１０（Ａ）に示すようにスタックポインタ移動命令は、操作機能がＳＰ１４に格納されたスタックポインタへの移動情報の加算及び減算であることを示すオペコード６３２（６ビット）、即値で指定された即値移動情報６３４（１０ビット）を含み、１６ビットのオブジェクトコードを有している。前記オペコード６３２はＳＰ１４を操作対象とするスタックポインタ移動命令であることを示す共通のコードと加算及び減算の別に応じて与えられる異なるコードとを含んでいる。従ってオペコードによりＳＰ１４に格納されたスタックポインタを操作対象とすることがわかるため、オブジェクトコードのオペランドにスタックポインタに関する情報を必要としない。即値移動情報６３４はスタックポインタに加算又は減算を行うオフセット値を生成するための情報である。

【０１４９】図１０（Ｂ）はスタックポインタとして汎用レジスタを用いた場合に使用する加算及び減算命令（以下汎用演算命令という）のオブジェクトコードのビットフィールド６４０の例を示している。

【０１５０】図１０（Ｂ）に示すように汎用演算命令は、操作機能が汎用レジスタの値への即値演算情報の加算及び減算であることを示すオペコード６４２（６ビット）、即値で指定された即値演算情報６４４（１０ビット）と操作対象となる汎用レジスタを特定するレジスタ番号６４６（４ビット）とを含み、２０ビットのオブジェクトコードを有している。汎用マイクロコンピュータでは命令長は８ビット単位なので２４ビット若しくは３２ビット命令になる。

【０１５１】図１０（Ａ）（Ｂ）は、いずれもスタックポインタに即値を加算及び減算する処理を行う場合に使用する命令のオブジェクトコードであるが、同図に示すように、スタックポインタ移動命令は汎用演算命令に比べて短いオブジェクトコードで記述することができる。

【０１５２】以下、（９）のＳＰに対する即値加算命令（以下加算スタックポインタ移動命令という）と、（１

０）のＳＰに対する即値減算命令（以下減算スタックポインタ移動命令という）を実行するための構成及び実行時の動作について説明する。

【０１５３】まず図１を用いてこれらの命令を実行するために必要なハードウエア構成について説明する。これらの命令は外部のメモリ（ＲＯＭ）５２よりＩ＿ＤＡＴＡ＿ＢＵＳ９４を介して伝送され、ＣＰＵ１０の命令デコーダ２０に入力される。該命令デコーダ２０で、命令が解読され命令の実行に必要な図示しない各種信号が出力される。また前記即値生成器２２は、前記即値移動情報６３４の１０ビットを２ビット左論理シフトし、ゼロ拡張してＰＡ＿ＢＵＳ７２に出力する。ＳＰ１４はスタックポインタを格納しており、この値はＸＡ＿ＢＵＳ７８に出力される。ＸＡ＿ＢＵＳ７８はＡＬＵ４０の入力となるＰＢ＿ＢＵＳ７４に接続されている。ＡＬＵ４０のもう一方の入力は、即値生成器２２の出力であるＰＡ＿ＢＵＳ７２に接続されている。ＡＬＵ４０の出力は、ＷＷ＿ＢＵＳ７６に接続されている。ＷＷ＿ＢＵＳ７６はＳＰの入力に接続されている。

【０１５４】まず加算スタックポインタ移動命令の実行時の動作について説明する。

【０１５５】加算スタックポインタ移動命令が実行されると、ＳＰ１４に格納されたスタックポインタの値と、前記即値移動情報６３４に基づき即値生成器２２が生成した移動即値が加算されて新たなスタックポインタが生成され、その値がＳＰ１４に格納される。

【０１５６】図１１は加算スタックポインタ移動命令の動作を説明するためのフローチャート図である。

【０１５７】前記命令の実行の最初にＳＰ１４に格納されているスタックポインタがＸＡ＿ＢＵＳ７８に出力される（ステップ２５０）。そして前記ＸＡ＿ＢＵＳ７８上のデータはＰＢ＿ＢＵＳ７４に出力される（ステップ２５２）。また、即値生成器２２が即値移動情報に基づき生成した移動即値ｉｍｍがＰＡ＿ＢＵＳ７２に出力される（ステップ２５４）。ＡＬＵ４０は、前記ＰＢ＿ＢＵＳ７４上の値と前記ＰＡ＿ＢＵＳ７２上の値を加算し、結果をＷＷ＿ＢＵＳ７６に出力する（ステップ２５６）。そして、ＷＷ＿ＢＵＳ７６上の値がＳＰ１４に入力される（ステップ２５８）。

【０１５８】次に減算スタックポインタ移動命令の実行時の動作について説明する。

【０１５９】減算スタックポインタ移動命令が実行されると、ＳＰ１４に格納されたスタックポインタの値から、前記即値移動情報６３４に基づき即値生成器２２が生成した移動即値が減算されて新たなスタックポインタが生成され、その値がＳＰ１４に格納される。

【０１６０】図１２は減算スタックポインタ移動命令の動作を説明するためのフローチャート図である。

【０１６１】前記命令の実行の最初にＳＰ１４に格納されているスタックポインタがＸＡ＿ＢＵＳ７８に出力される（ステップ２６０）。そして前記ＸＡ＿ＢＵＳ７８上のデータはＰＢ＿ＢＵＳ７４に出力される（ステップ２６２）。また、即値生成器２２が即値移動情報に基づき生成した移動即値ｉｍｍがＰＡ＿ＢＵＳ７２に出力される（ステップ２６４）。ＡＬＵ４０は、前記ＰＢ＿ＢＵＳ７４上の値から前記ＰＡ＿ＢＵＳ７２上の値を減算し、結果をＷＷ＿ＢＵＳ７６に出力する（ステップ２６６）。そして、ＷＷ＿ＢＵＳ７６上の値がＳＰ１４に入力される（ステップ２６８）。

【０１６２】（７）分岐命令

図１３は、ｃａｌｌ命令とｒｅｔ命令によるプログラムの実行の制御を説明するための図である。図１３に示すように、ＭＡＩＮルーチン３００において、サブルーチンＳＵＢ３１０へ分岐するためのｃａｌｌ命令が実行されると（３０２）、制御がサブルーチンＳＵＢ３１０に渡る。サブルーチンの最後には、ｒｅｔ命令（３１２）が書かれており、この命令が実行されると、ＭＡＩＮルーチン３００の前記ｃａｌｌ命令（３０２）の次の命令（３０４）に戻る。従って、図１３に示す場合①②③の順でプログラムが実行されることになる。このようにサブルーチンＳＵＢ３１０での実行が終わるとＭＡＩＮルーチン３００に戻り、前記ｃａｌｌ命令（３０２）の次の命令（３０４）から実行するため、サブルーチンＳＵＢ３１０に分岐する際、どこかに戻り先のアドレスを記憶しておくことが必要となる。このため、ｃａｌｌ命令等のサブルーチンへ分岐する分岐命令実行の際には、図４で説明したように戻り先アドレスをスタックに待避する処理が行われ、ｒｅｔ命令等のサブルーチンから戻る分岐命令の実行の際には前記スタックから戻り先アドレスをプログラムカウンタに戻す処理（以下プログラムカウンタの待避及び復旧の処理という）が行われる。

【０１６３】従来のＲＩＳＣ方式のＣＰＵでは、前記プログラムカウンタの待避及び復旧の処理をソフトウエア的に実現していたため、ｃａｌｌ命令等の分岐命令を実行する際には、これらの処理を実行するためオブジェクトコード（アセンブラ命令）も必要であった。例えばｃａｌｌ命令を実行する際には、スタックポインタをワードサイズ（４）だけデクリメント（－４）して、プログラムカウンタの値に基づきｃａｌｌ命令の次の命令のアドレスをスタックに格納するためのオブジェクトコード（アセンブラ命令）が必要であった。

【０１６４】しかし本実施例のＣＰＵは、前記ｃａｌｌ命令やｒｅｔ命令が実行されると、前記プログラムカウンタの待避及び復旧の処理も一緒に行うようなハードウエア構成を有している。従って、前記ｃａｌｌ命令やｒｅｔ命令と別に、前記プログラムカウンタの待避及び復旧の処理を記述するオブジェクトコード（アセンブラ命令）を必要としない。本実施例のＣＰＵでは、このような前記プログラムカウンタの待避及び復旧の処理を一命令で実行するために、スタックポインタ専用命令の一つ

である分岐命令として次に示す命令セットを用意している。

【０１６５】ｃａｌｌ　sign9　　…（１１）
ｃａｌｌ　%Ｒｂ　　…（１２）
ｒｅｔ　　…（１３）
ｒｅｔｉ　　…（１４）
ｒｅｔｄ　　…（１５）
ｉｎｔ　imm2　　…（１６）
ｂｒｋ　　…（１７）

（１１）～（１７）は命令コードをアセンブラで記述したものである。（１１）はＰＣ相対サブルーチンコール命令で、プログラムカウンタＰＣをベースアドレスとして、オペランドで指定されたディスプレースメント情報sign9に基づき分岐先アドレスを相対的に指定して分岐するコール命令である。（１２）はレジスタ間接サブルーチンコール命令であり、オペランドで指定されたレジスタに格納されている分岐先アドレスに分岐するコール命令である。（１３）はサブルーチンからのリターン命令である。（１４）は割り込み又は例外処理ルーチンからのリターン命令である。（１５）はデバッグ処理ルーチンからのリターン命令である。（１６）はソフトウエア割り込み命令である。（１７）はソフトウエア・デバッグ割り込み命令である。

【０１６６】図１４は、前記（１１）のＰＣ相対サブルーチンコール命令のビットフィールド６５０である。図１４に示すようにＰＣ相対サブルーチンコール命令は、操作機能がプログラムカウンタをベースアドレスとして、分岐先アドレスを相対的に指定してサブルーチンへ分岐するコール命令であることを示すオペコード６５２（８ビット）、即値で指定されたディスプレースメント情報sign9（８ビット）６５４とを含み、１６ビットのオブジェクトコードを有している。前記８ビットの即値は、実行時に１ビット左に論理シフトされた後、サイン拡張される。

【０１６７】本実施例のＣＰＵでは、ｃａｌｌ命令実行には図１４に示すオブジェクトコードのみで、プログラムカウンタのスタックへの待避も実行することが出来る。

【０１６８】以下、サブルーチンへ分岐する命令として（１１）のＰＣ相対サブルーチンコール命令、サブルーチンからリターンする命令として（１３）のリターン命令を例にとりこれらの命令を実行するための構成及び実行時の動作について説明する。

【０１６９】まず図１を用いてこれらの命令を実行するために必要なハードウエア構成について説明する。これらの命令は外部のメモリ（ＲＯＭ）５２よりＩ＿ＤＡＴＡ＿ＢＵＳ９４を介して伝送され、ＣＰＵ１０の命令デコーダ２０に入力される。該命令デコーダ２０で、命令が解読され命令の実行に必要な図示しない各種信号が出力される。また前記即値生成器２２は、前記ディスプレースメント情報６５４を１ビット左に論理シフトした後サイン拡張して３２ビットの即値ディスプレースメントｉｍｍを生成し、ＰＢ＿ＢＵＳ７４に出力する。ＳＰ１４はスタックポインタを格納しており、この値はアドレス加算器３０の入力に接続されたＸＡ＿ＢＵＳ７８に出力できる。アドレス加算器３０のもう一方の入力は、即値生成器２２の出力であるＰＢ＿ＢＵＳ７４に接続されている。アドレス加算器３０の出力（ＡＤＤＲ）は、ＩＡ信号線８２を介して外部のＩ＿ＡＤＤＲ＿ＢＵＳ９２に接続されている。

【０１７０】また、Ｉ＿ＡＤＤＲ＿ＢＵＳ９２及びＩ＿ＤＡＴＡ＿ＢＵＳ９４は命令のオブジェクトコードが格納されたＲＯＭ５２に接続されており、バスコントロールユニット（ＢＣＵ）６０は、ＣＰＵから出力される各種リクエスト信号（外部バスに出力された信号等）に基づき、メモリ（ＲＯＭ）５２から前記命令のオブジェクトコードを読み出すＲＥＡＤ制御信号を出力する。

【０１７１】まずＰＣ相対サブルーチンコール命令の実行時の動作について説明する。

【０１７２】ＰＣ相対サブルーチンコール命令が実行されると、図４で説明したように、ＰＣ１２に格納されたプログラムカウンタの値がスタックに待避され、ＳＰ１４に格納されたスタックポインタの値がワードサイズ（４）だけディクリメントされる。そしてＰＣ１２にプログラムカウンタと前記３２ビットの即値ディスプレースメントを加算して得られた分岐先アドレスがセットされる。

【０１７３】図１５はＰＣ相対サブルーチンコール命令の動作を説明するためのフローチャート図である。

【０１７４】前記命令の実行の最初にＳＰ１４に格納されているスタックポインタがＸＡ＿ＢＵＳ７８に出力される（ステップ２７０）。これはアドレス加算器の一方の入力となり、即値生成器２２が生成したconstantデータ（－４）がアドレス加算器３０の他方の入力となる。そしてアドレス加算器３０で前記ＸＡ＿ＢＵＳ７８上のスタックポインタの値と－４を加算して戻りアドレスを格納するスタックへの書き込みアドレス生成され、ＷＷ＿ＢＵＳ７６に出力される。また、前記書き込みアドレスはＤＡ信号線８４を介してＤ＿ＡＤＤＲ＿ＢＵＳ９６に出力される。（ステップ２７２）。また、ＰＣインクリメンタ４４ではＰＣ１２に格納されたプログラムカウンタの値が＋２されて戻りアドレスが生成されて、ＤＯＵＴ信号線８８を介してＤ＿ＤＡＴＡ＿ＢＵＳ９８に出力される（ステップ２７４）。そしてＣＰＵからＢＣＵ６０へのデータ書き出しリクエスト信号がアクティブになり、外部メモリのライトサイクルを実行する（ステップ２７６）。即ち前記ＢＣＵ６０は該リクエスト信号に基づき、前記書き込みアドレスをメモリアドレスとしてメモリに設けられたスタックに前記戻りアドレスが格納される。

【０１７５】そして、ＷＷ＿ＢＵＳ７６上の値がＳＰ１４に出力される（ステップ２７８）。すなわちスタックポインタの値が－４した値に更新される。

【０１７６】次にＰＣ１２に格納されているプログラムカウンタがＸＡ＿ＢＵＳ７８に出力される（ステップ２８０）。また、前記即値生成器２２は、命令のオブジェクトコードに含まれている前記ディスプレースメント情報６５４を１ビット左に論理シフトした後サイン拡張して３２ビットの即値ディスプレースメントｉｍｍを生成し、ＰＢ＿ＢＵＳ７４に出力する。前記アドレス加算器３０はＸＡ＿ＢＵＳ７８上のプログラムカウンタとＰＢ＿ＢＵＳ７４上の即値ディスプレースメントｉｍｍを加算し分岐アドレス（ＡＤＤＲ）を生成し、ＩＡ信号線８２を介して、Ｉ＿ＡＤＤＲ＿ＢＵＳ９２に出力する（ステップ２８２）。そしてＣＰＵからＢＣＵ６０への命令読み出しリクエスト信号がアクティブとなり、外部メモリ（ＲＯＭ）５２のリードサイクルを実行し、分岐先の命令のオブジェクトコードを読み出す（ステップ２８４）。

【０１７７】次にｒｅｔ命令の実行時の動作について説明する。

【０１７８】ｒｅｔ命令が実行されると、図４で説明したように、スタックに待避されていたプログラムカウンタの値がＰＣ１２に復帰され、ＳＰ１４に格納されたスタックポインタの値がワードサイズ（４）だけインクリメントされる。

【０１７９】図１６はｒｅｔ命令の動作を説明するためのフローチャート図である。

【０１８０】前記命令の実行前には、ＳＰ１４に格納されたスタックポインタは呼ばれたルーチンへの戻り先アドレスが格納されたスタックのアドレスをさしている。

【０１８１】前記命令の実行の最初にＳＰ１４に格納されているスタックポインタがＸＡ＿ＢＵＳ７８に出力される（ステップ２９０）。そしてＸＡ＿ＢＵＳ７８上のスタックポインタはＤ＿ＡＤＤＲ＿ＢＵＳ９６に出力される（ステップ２９２）。そしてＣＰＵからＢＣＵ６０へのデータ読み込みリクエスト信号がアクティブになり、外部メモリのリードサイクルを実行する。即ち前記ＢＣＵ６０は該リクエスト信号に基づき、前記スタックポインタをメモリアドレスとしてメモリに設けられたスタックから前記戻りアドレスを読み出す。メモリ（ＲＡＭ）５０から読み出された前記戻り先アドレスはＤ＿ＤＡＴＡ＿ＢＵＳ９４からＤＩＮ信号線８６を介してＣＰＵ内部にとりこまれ、さらにＤＩＮ信号線からＩＡ信号線８２を介してＩ＿ＡＤＤＲ＿ＢＵＳ９２に出力される（ステップ２９４）。そしてＣＰＵからＢＣＵ６０への命令読み出しリクエスト信号がアクティブとなり、外部メモリ（ＲＯＭ）５２のリードサイクルを実行し、戻り先アドレスの命令のオブジェクトコードを読み出す（ステップ２９６）。

【０１８２】そして、ＸＡ＿ＢＵＳ７８上のスタックポインタの値は、前記アドレス加算器３０の一方の入力となる。また即値生成器２２が生成したconstantデータ（＋４）がアドレス加算器３０の他方の入力となる。そしてアドレス加算器３０で前記ＸＡ＿ＢＵＳ７８上のスタックポインタの値と＋４を加算して、戻り先のルーチンが確保したスタック領域の先頭エリアのアドレスが生成され、ＷＷ＿ＢＵＳ７６に出力される（ステップ２９８）。そして、ＷＷ＿ＢＵＳ７６上の前記アドレス（戻り先のルーチンが確保したスタック領域の先頭エリアのアドレス）がＳＰ１４に出力される（ステップ３００）。

【０１８３】（８）連続プッシュ命令（ｐｕｓｈｎ）、連続ポップ命令（ｐｏｐｎ）の説明

前述したように、最近の特にＲＩＳＣ方式のＣＰＵは、性能を高めるため内部に多くの汎用レジスタを持ち、メモリにアクセスすることなくＣＰＵ内部で高速に多くの処理をおこなうよう構成されている。本実施例でも内部に１６本の汎用レジスタをもち、処理の高速化を図っている。しかし、このように内部レジスタを多く持つと，割り込み処理やサブルーチンコールの時のレジスタ退避と復旧の処理の際、退避すべきレジスタ数が多くなる。

【０１８４】このようなレジスタの待避や復旧を行う場合、従来は、番地部で指定した内容をスタックに格納するｐｕｓｈ命令や、スタックの内容をレジスタに取り出すｐｏｐ命令を用いていた。ここで一般的なｐｕｓｈ命令及びｐｏｐ命令時の動きを説明する。

【０１８５】図１７（Ａ）（Ｂ）はｐｕｓｈ命令実行時の動きを模式的に示した図であり、図１８（Ａ）（Ｂ）はｐｏｐ命令実行時の動きを模式的に示した図である。図１７（Ａ）（Ｂ）及び図１８（Ａ）（Ｂ）を用いて、複数の汎用レジスタとスタック間でデータの転送を行う場合の動きを説明する。

【０１８６】図１７（Ａ）は、汎用レジスタＲ１の内容をスタックに書き出す命令である’ｐｕｓｈ　Ｒ１’が実行された場合の動きを示している。該命令の実行時には、ＳＰ１４の内容は現在の値から４がひかれた値に更新される（図１７（Ａ）に示すように１０００が９９６に更新される）。そして、更新されたＳＰ１４のスタックポインタが示すメモリアドレスである９９６に汎用レジスタＲ１の内容ａが書き込まれる。

【０１８７】図１７（Ｂ）は、さらに汎用レジスタＲ２の内容をスタックに書き出す命令である’ｐｕｓｈ　Ｒ２’が実行された場合の動きを示している。該命令の実行時には、ＳＰ１４の内容は現在の値から４がひかれた値に更新される（図１７（Ｂ）に示すように９９６が９９２に更新される）。そして、更新されたＳＰ１４のスタックポインタが示すメモリアドレスである９９２に汎用レジスタＲ２の内容ｂが書き込まれる。

【０１８８】図１８（Ａ）は、スタックの内容を汎用レ

ジスタＲ２に取り出す命令である’ｐｏｐ　Ｒ２’が実行された場合の動きを示している。該命令の実行時には、ＳＰ１４のスタックポインタが示しているメモリアドレス９９２に格納されている内容ｂがとりだされて汎用レジスタＲ２に格納される。そしてＳＰ１４の内容は現在の値に４が足された値に更新される（図１８（Ａ）示すように実行前９９２であったのが実行後９９６に更新される）。

【０１８９】図１８（Ｂ）は、更にスタックの内容を汎用レジスタＲ１に取り出す命令である’ｐｏｐ　Ｒ１’が実行された場合の動きを示している。該命令の実行時には、ＳＰ１４のスタックポインタが示しているメモリアドレス９９６に格納されている内容ａがとりだされて汎用レジスタＲ１に格納される。そしてＳＰ１４の内容は現在の値に４が足された値に更新される（図１８（Ｂ）に示すように実行前９９６であったのが実行後１０００に更新される）。

【０１９０】このように従来は、複数の汎用レジスタとスタックでデータの転送を行う場合、ｐｕｓｈ命令やｐｏｐ命令を複数回繰り返して実行することが必要であった。ｐｕｓｈ命令やｐｏｐ命令は１回の命令で１本のレジスタしか操作することができなかったからである。

【０１９１】従って多数のレジスタに対してスタックへの待避やスタックからの復旧処理を行う場合、命令数の増加によりオブジェクトコードのサイズの増大を招いていた。また、プログラムの実行ステップも多くなり、プログラムの実行時間や処理動作の遅延を招いていた。

【０１９２】そこで 本実施例のＣＰＵでは、次に示す命令セットを用意している。

【０１９３】ｐｕｓｈｎ　　％Ｒｓ　　…（１８）
ｐｏｐｎ　　％Ｒｄ　　…（１９）

（１８）は連続プッシュ命令のアセンブラの記述を示しており、％ＲｓからＲ０までのｎ個（ｎは１から１６の自然数）の汎用レジスタの内容を連続的にスタックへプッシュする命令である。（１９）は連続ポップ命令のアセンブラの記述を示しており、スタックからｎ個（ｎは１から１６の自然数）のワードデータを連続的に％ＲｄからＲ０までの汎用レジスタへポップする命令である。ｐｕｓｈｎ、ｐｏｐｎ命令は、いずれもオペコードとオペランドからなる。％ＲｓはｐｕｓｈｎＲ命令のオペランドでレジスタ％ＲｓからＲ０までをスタックに書き込む場合のＲｓのレジスタ番号を示している。％ＲｄはｐｏｐｎＲ命令のオペランドでＲ０から％Ｒｄ迄のレジスタにスタックからデータを持ってくる場合のＲｄのレジスタ番号を示している。

【０１９４】図１９にｐｕｓｈｎ、ｐｏｐｎ命令のビットマップを示す。下位の４ビットのフィールドに％Ｒｓまたは％Ｒｄを示すコードが入り、１６本ある汎用レジスタの任意のレジスタが指定できる。特殊レジスタとスタックの間でデータの転送をするときは、汎用レジスタを介して行う。

【０１９５】図２０は連続プッシュ命令（ｐｕｓｈｎ）、連続ポップ命令（ｐｏｐｎ）を実行するためのハードウェア構成を説明するためのブロック図である。図１から連続プッシュ命令（ｐｕｓｈｎ）及び連続ポップ命令（ｐｏｐｎ）の説明に必要な部分を取り出して、説明に必要な部分を追加した構成になっている。図１と同一部分を指すものについて同一の番号を付している。図中１１はＲ０からＲ１５の１６本の汎用レジスタである。汎用レジスタ１１はデータバス（Ｄ＿ＤＡＴＡ＿ＢＵＳ）９８とデータの入出力を行う。またレジスタを選択するレジスタ選択アドレス信号５４はコントロール回路ブロック４５から来る。１４はＳＰでスタックポインタが格納されている。ＳＰ１４の値はアドレスバス（Ｄ＿ＡＤＤＲ＿ＢＵＳ）９６と３２ビットのアドレス加算器３０の入力に接続する内部アドレスバス（ＸＡ＿ＢＵＳ）７８に出力できる。アドレス加算器３０のもう一方の入力はコントロール回路ブロック４５からの出力されるオフセット信号２４に接続されている。アドレス加算器３０の出力はラッチ（Ａｄｄ＿ＬＴ）３２でラッチされ、さらにＸＡ＿ＢＵＳ７８またはＷＷ＿ＢＵＳ７６に出力される。ＷＷ＿ＢＵＳ７６はＳＰ１４の入力に接続されている。コントロール回路ブロック４５の中には４ビットのカウンタ（ｃｏｕｎｔｘ）４６があり、転送するレジスタ数をカウントする。又、コントロール回路ブロック４５中には図２０では示されていないがインストラクションレジスタがあり、ｐｕｓｈｎ、ｐｏｐｎ命令のオペランド％Ｒｓ、％Ｒｄを保持するとともに、インストラクションデコーダによって各命令に応じた制御信号を出力する。図中６０はバスコントロールユニット（ＢＣＵ）で外部のスタックエリアを含むメモリ（ＲＡＭ）５０とのデータの入出力を制御し、ＲＥＡＤ、ＷＲＩＴＥ制御信号を出力する。

【０１９６】まず連続プッシュ命令（ｐｕｓｈｎ）の実行時の動作について説明する。

【０１９７】連続プッシュ命令（ｐｕｓｈｎ）が実行されると、図４で説明したように、％Ｒｓのレジスタ番号の汎用レジスタから汎用レジスタＲ０までの汎用レジスタに格納された内容が連続的にスタックにプッシュされる。

【０１９８】図２１はｐｕｓｈｎ命令の動作を説明するためのフローチャート図である。

【０１９９】ｐｕｓｈｎ命令の実行の最初にオフセット信号２４のoffsetは－４になる。カウンタ（ｃｏｕｎｔｘ）４６はゼロにクリアされ、ＳＰ１４に格納されたスタックポインタ値がＸＡ＿ＢＵＳ７８に出力される（ステップ１００）。

【０２００】次にアドレス加算器３０はＸＡ＿ＢＵＳ７８上の値と－４を加算し結果をラッチ（Ａｄｄ＿ＬＴ）３２に入れる（ステップ１０１）。

【０２０１】ラッチ（Ａｄｄ＿ＬＴ）３２の値はアドレスバスＤ＿ＡＤＤＲ＿ＢＵＳ９６に出力される。コントロール回路ブロック４５ではインストラクションレジスタの下位４ビットに保持された％Ｒｓとカウンタ（ｃｏｕｎｔｘ）４６の差を計算し結果をレジスタ選択アドレス信号５４に出力する。５４によって選択されたレジスタをデータバス（Ｄ＿ＤＡＴＡ＿ＢＵＳ）９８にのせ、外部メモリ（ＲＡＭ）５０の書き込み動作をする（ステップ１０２）。

【０２０２】ステップ１０３では、カウンタ（ｃｏｕｎｔｘ）４６と％Ｒｓを比較する。等しいときはレジスタの外部メモリへの書き込みは完了しており、ラッチ（Ａｄｄ＿ＬＴ）３２の値をＷＷ＿ＢＵＳ７６を通じてＳＰ１４に書き込んでｐｕｓｈｎの実行を終わる（ステップ１０４）。

【０２０３】等しくないときはカウンタ（ｃｏｕｎｔｘ）をプラス１し、ラッチ（Ａｄｄ＿ＬＴ）３２の値をＸＡ＿ＢＵＳ７８に出力し、ステップ１０１以降の処理を繰り返す（ステップ１０５）。

【０２０４】次に連続ポップ命令（ｐｏｐｎ）の実行時の動作について説明する。

【０２０５】連続ポップ命令（ｐｏｐｎ）が実行されると、図４で説明したように、スタックの内容が汎用レジスタＲ０から％Ｒｄのレジスタ番号の汎用レジスタに連続的にプッシュされる。

【０２０６】図２２は、ｐｏｐｎ命令の動作を説明するためのフローチャート図である。

【０２０７】ｐｏｐｎ命令の最初にオフセット信号２４は＋４になる。カウンタ（ｃｏｕｎｔｘ）４６はゼロクリアされ、ＳＰ１４のスタックポインタの値がＸＡ＿ＢＵＳ７８とアドレスバス（Ｄ＿ＡＤＤＲ＿ＢＵＳ）９６に出力される（ステップ１１０）。

【０２０８】次にアドレス加算器３０はＸＡ＿ＢＵＳ７８上の値と＋４を加算し結果をラッチ（Ａｄｄ＿ＬＴ）３２に入れる（ステップ１１１）。

【０２０９】次に外部メモリのリードサイクルを実行する。リードされたデータはデータバス（Ｄ＿ＤＡＴＡ＿ＢＵＳ）９８を通じて汎用レジスタ１１に書き込まれる。このときコントロール回路ブロック４５ではカウンタ（ｃｏｕｎｔｘ）４６をレジスタ選択アドレス信号５４に出力する（ステップ１１２）。

【０２１０】ステップ１１３では、カウンタ（ｃｏｕｎｔｘ）４６と％Ｒｓを比較する。等しいときはレジスタの外部メモリへの書き込みは完了しており、ラッチ（Ａｄｄ＿ＬＴ）３２の値をＷＷ＿ＢＵＳ７６を通じてＳＰ１４に書き込んでｐｏｐｎの実行を終わる（ステップ１１４）。

【０２１１】等しくないときはカウンタ（ｃｏｕｎｔｘ）をプラス１し、ラッチ（Ａｄｄ＿ＬＴ）３２の値をＸＡ＿ＢＵＳ７８とアドレスバス（Ｄ＿ＡＤＤＲ＿ＢＵＳ）９６に出力し、ステップ１１１以降の処理を繰り返す（ステップ１１５）。

【０２１２】このように’ｐｕｓｈｎ　％Ｒｓ’で％ＲｓからＲ０までのレジスタをスタックに書き込め，また’ｐｏｐｎ　％Ｒｄ’でスタックから必要な個数のデータをＲ０から％Ｒｄまでのレジスタに戻すことが出来る。たとえば、’ｐｕｓｈｎ％Ｒ３’の実行によりレジスタＲ３からＲ０までを一命令でプッシュできる。

【０２１３】ここにおいてレジスタの使い方に一つの制約を加えることで、更に有効な効果が発生する。即ち、レジスタの待避や復旧は割り込み処理やサブルーチンコール時等のプログラムが他のルーチンに分岐するときに特に必要になるが、このとき他の割り込みルーチンまたはサブルーチン等の呼び出されるルーチンではレジスタをＲ０から順に使って行くことが好ましい。このようにすると、Ｒ０からＲｄまたは％Ｒｓに退避の必要のないレジスタが含まれず、ｐｕｓｈｎ命令及びｐｏｐｎ命令を用いて、効率的にレジスタの待避又は復帰を行うことができる。本実施例ではレジスタはどれも同じ機能を持っており、レジスタの使い方と命令には何の制約も無いので、このような制約は何の問題もなく満たすことが出来る。

【０２１４】従って本実施例のｐｕｓｈｎ命令及びｐｏｐｎ命令を用いることにより、レジスタからメモリ中のスタックへの退避、またスタックからレジスタへのデータの復旧がｐｕｓｈｎ、ｐｏｐｎの各々一命令で実行できる。このためオブジェクトコードサイズやプログラム実行ステップを最少にし、かつ実行サイクルを一回の命令フェッチと必要な回数のデータの転送だけですませ、最少のサイクルで行うことが出来る。これにより割り込み処理ルーチンやサブルーチンの処理も高速化を図ることができる。

【０２１５】一方、これを実現する構成要素はカウント手段と簡単なシーケンス制御のみであり、少ないゲート数で実現出来、ワンチップのマイクロコンピュータに適したものである。

【０２１６】また、本実施例ではスタックポインタ専用のレジスタＳＰ１４を用いた場合の連続プッシュ命令（ｐｕｓｈｎ）、連続ポップ命令（ｐｏｐｎ）を実行させるための構成について説明したが、スタックポインタとして汎用レジスタを使用する場合にも適用可能である。

【０２１７】（実施例２）図２３は、本実施例のマイクロコンピュータのハードウエアブロック図である。

【０２１８】該マイクロコンピュータ２は、３２ビットマイクロコントローラであり、ＣＰＵ１０とＲＯＭ５２とＲＡＭ５０、高周波発振回路９１０、低周波発振回路９２０、リセット回路９３０、プリスケーラ９４０、１６ビットプログラマブルタイマ９５０、８ビットプログラマブルタイマ９６０、クロックタイマ９７０、インテ

リジェントＤＭＡ９８０、高速ＤＭＡ９９０、割り込みコントローラ８００、シリアルインターフェース８１０、バスコントロールユニット（ＢＣＵ）６０、Ａ／Ｄ変換器８３０、Ｄ／Ａ変換器８４０、入力ポート８５０、出力ポート８６０、Ｉ／Ｏポート８７０、及びそれらを接続する各種バス９２、９４、９６、９８、各種ピン８９０等を含む。

【０２１９】前記ＣＰＵ１０は、スタックポインタ専用レジスタであるＳＰを有し、前述した各種のスタックポインタ専用命令の解読、実行を行う。該ＣＰＵ１０は、前述した、実施例１の構成を有しており、前記解読手段、前記実行手段として機能する。

【０２２０】従って本実施例のマイクロコンピュータは、スタックポインタを取り扱う処理を、短い命令長で効率よく記憶し、実行することができる。

【０２２１】また、レジスタ退避やレジスタ復旧の処理を効率よく記憶し、割り込み処理及びサブルーチンコール・リターンの処理を高速に行うことができる。

【０２２２】本発明のマイクロコンピュータは例えばプリンター等のパソコン周辺機器や、携帯機器等の各種の電子機器に適用可能である。この様にすると、簡単な構成でメモリの使用効率がよく高速に処理の行える情報処理回路を内蔵することができるため、安価で高機能な電子機器を提供することが出来る。

【０２２３】なお本発明は、上記実施例で説明したものに限らず、種々の変形実施が可能である。

【０２２４】

【図面の簡単な説明】

【図１】本実施例のＣＰＵの回路構成の概略を説明するための図である。

【図２】本実施例のＣＰＵの持つレジスタセットを示す。

【図３】スタックポインタの一般的な動作について説明するための図である。

【図４】スタックポインタ専用命令の使用例とメモリにもうけられたスタックの使用状態及びスタックポインタの状態について説明するための図である。

【図５】メモリ上のオート変数をレジスタに転送する処理を説明するための図である。

【図６】図６（Ａ）（Ｂ）は、ＳＰ相対ロード命令及び汎用ロード命令のビットフィールドを示した図である。

【図７】ワードデータ読み出しのＳＰ相対ロード命令の動作を説明するためのフローチャート図である。

【図８】ワードデータ書き込みのＳＰ相対ロード命令の動作を説明するためのフローチャート図である。

【図９】図９（Ａ）～（Ｆ）は、複数のルーチンにわたってプログラムが実行される場合の各ルーチンによるメモリ上のスタックの使用状態及びスタックポインタの状態を説明するための図である。

【図１０】図１０（Ａ）（Ｂ）は、スタックポインタ移

動命令及び汎用演算命令のビットフィールドを示した図である。

【図１１】加算スタックポインタ移動命令の動作を説明するためのフローチャート図である。

【図１２】減算スタックポインタ移動命令の動作を説明するためのフローチャート図である。

【図１３】ｃａｌｌ命令とｒｅｔ命令によるプログラムの実行の制御を説明するための図である。

【図１４】ＰＣ相対サブルーチンコール命令のビットフィールドを示した図である。

【図１５】ＰＣ相対サブルーチンコール命令の動作を説明するためのフローチャート図である。

【図１６】ｒｅｔ命令の動作を説明するためのフローチャート図である。

【図１７】図１７（Ａ）（Ｂ）はｐｕｓｈ命令実行時の動きを模式的に示した図である。

【図１８】図１８（Ａ）（Ｂ）はｐｏｐ命令実行時の動きを模式的に示した図である。

【図１９】ｐｕｓｈｎ、ｐｏｐｎ命令のビットマップを示す。

【図２０】連続プッシュ命令（ｐｕｓｈｎ）、連続ポップ命令（ｐｏｐｎ）を実行するためのハードウェア構成を説明するためのブロック図である。

【図２１】ｐｕｓｈｎ命令の動作を説明するためのフローチャート図である。

【図２２】ｐｏｐｎ命令の動作を説明するためのフローチャート図である。

【図２３】本実施の形態のマイクロコンピュータのハードウエアブロック図である。

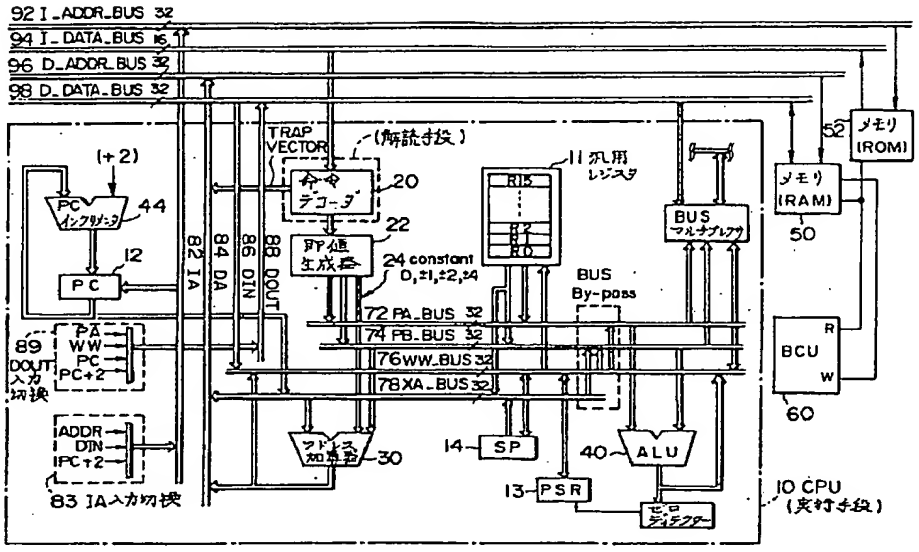
【符号の説明】

２　マイクロコンピュータ
１０　ＣＰＵ
１１　汎用レジスタ
１２　ＰＣ（プログラムカウンタ）
１３　ＰＳＲ（プロセッサステータスレジスタ）
２０　命令デコーダ
２２　即値生成器期
２４　オフセット信号
３０　アドレス加算器
３２　ラッチ（Ａｄｄ＿ＬＴ）
４０　ＡＬＵ
４４　ＰＣインクリメンタ
４５　コントロール回路ブロック
４６　４ビットカウンタ（ｃｏｕｎｔｘ）
５０　メモリ（ＲＡＭ）
５２　メモリ（ＲＯＭ）
５４　レジスタ選択アドレス信号
６０　バスコントロールユニット（ＢＣＵ）
７２、７４、７６、７８　内部バス
８２、８４、８６、８８　内部信号線

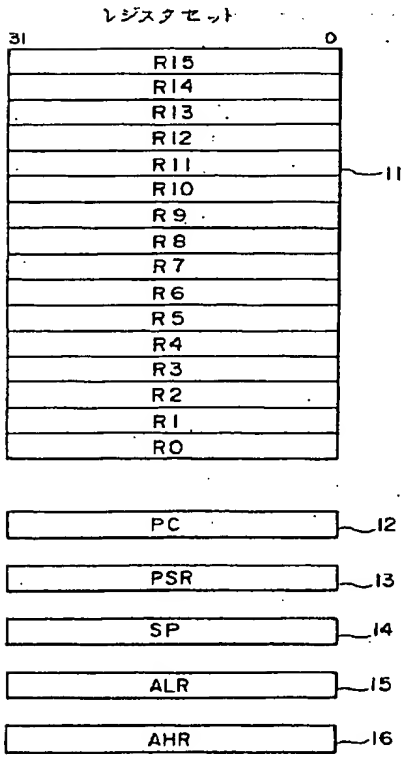
41　　42

９２、９４、９６、９８　外部バス

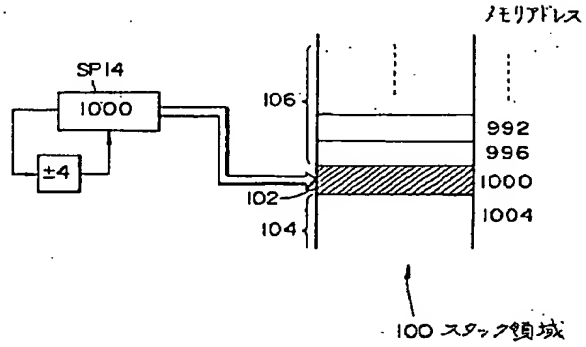
【図１】



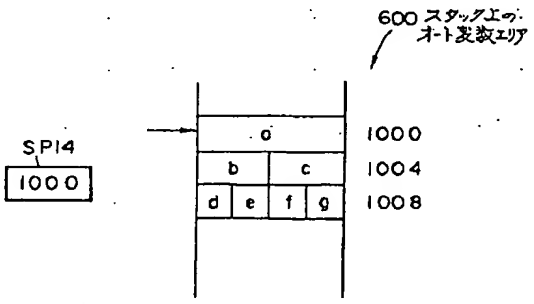
【図２】



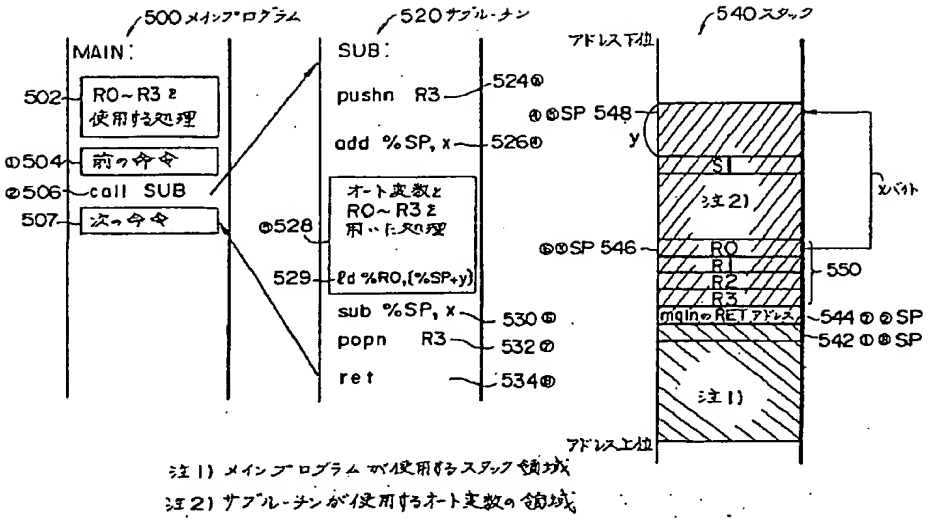
【図３】



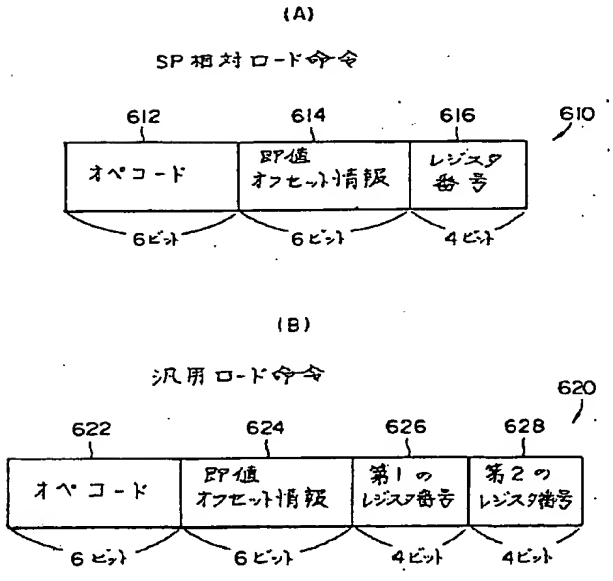
【図５】



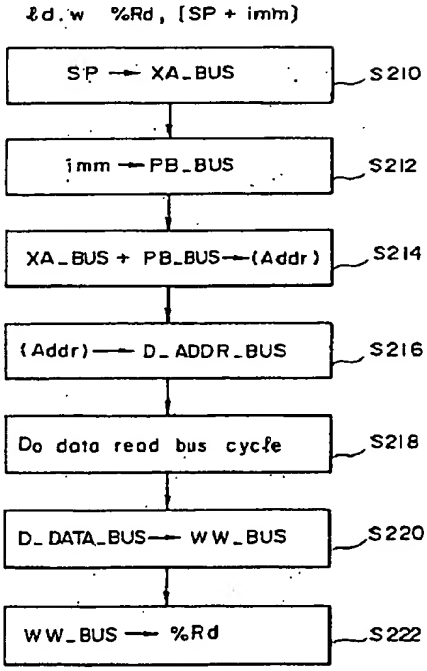
【図４】


500 メインプログラム
520 サブルーチン
540 スタック
MAIN:
502
R0～R3を使用する処理
①504 前の命令
②506 call SUB
507 次の命令
SUB:
pushn R3 524③
add %SP, x 526④
⑤528 オート変数とR0～R3を用いた処理
529 ld %R0,(%SP+y)
sub %SP, x 530⑥
popn R3 532⑦
ret 534⑧
アドレス下位
④⑧SP 548
y
S1
注2)
xバイト
⑤⑨SP 546
R0
R1
R2
R3
550
main の RET アドレス
544 ①②SP
542 ①③SP
注1)
アドレス上位
注1) メインプログラムが使用するスタック領域
注2) サブルーチンが使用するオート変数の領域

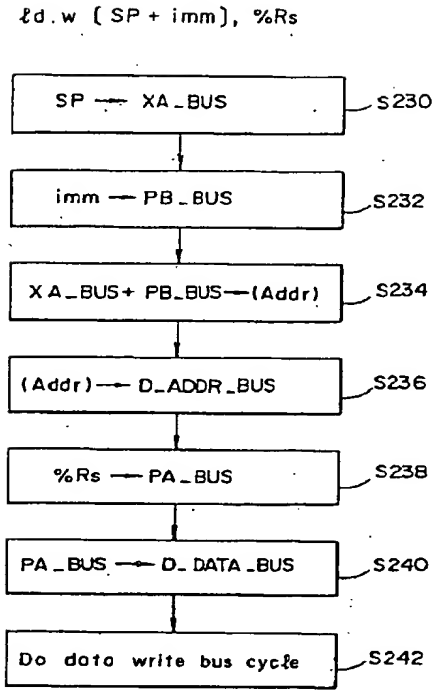
【図６】


(A)
SP相対ロード命令
612
614
616
610
オペコード
即値オフセット情報
レジスタ番号
6ビット
6ビット
4ビット
(B)
汎用ロード命令
620
622
624
626
628
オペコード
即値オフセット情報
第1のレジスタ番号
第2のレジスタ番号
6ビット
6ビット
4ビット
4ビット

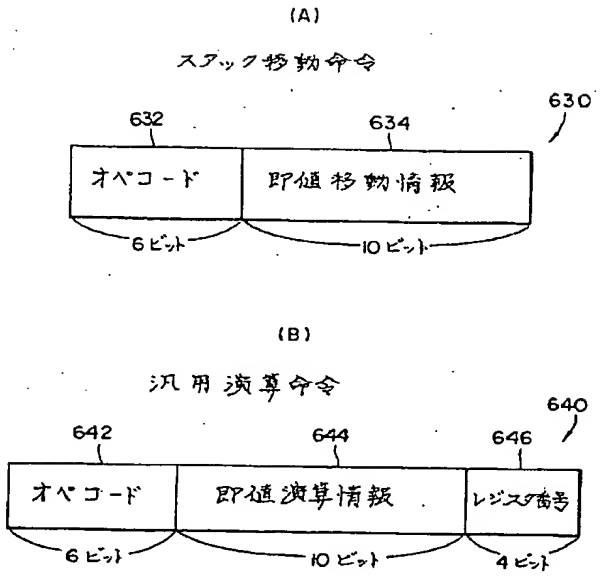
【図７】


ld.w %Rd, [SP + imm]
SP → XA_BUS
S210
imm → PB_BUS
S212
XA_BUS + PB_BUS → (Addr)
S214
(Addr) → D_ADDR_BUS
S216
Do data read bus cycle
S218
D_DATA_BUS → WW_BUS
S220
WW_BUS → %Rd
S222

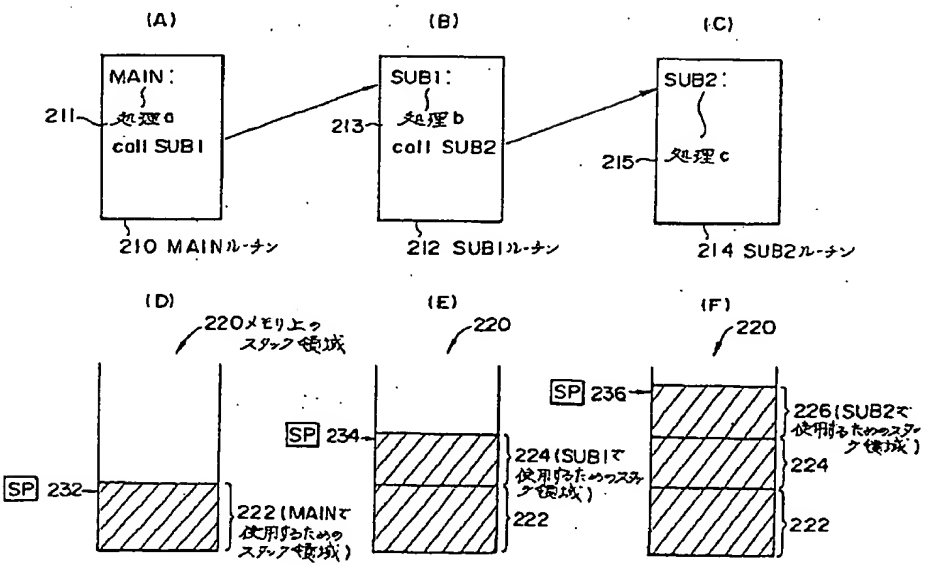
【図8】


ld.w [SP + imm], %Rs
SP → XA_BUS
S230
imm → PB_BUS
S232
XA_BUS+ PB_BUS→(Addr)
S234
(Addr)→ D_ADDR_BUS
S236
%Rs → PA_BUS
S238
PA_BUS → D_DATA_BUS
S240
Do data write bus cycle
S242

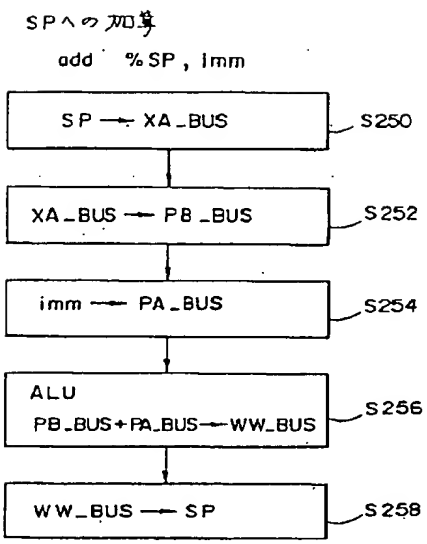
【図10】


(A)
スタック移動命令
630
632
634
オペコード
即値移動情報
6ビット
10ビット
(B)
汎用演算命令
640
642
644
646
オペコード
即値演算情報
レジスタ番号
6ビット
10ビット
4ビット

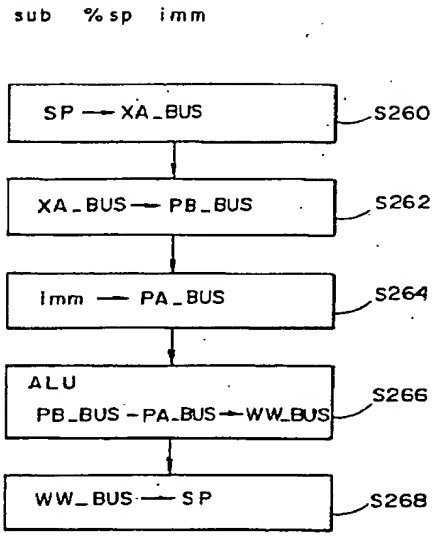
【図9】


(A)
(B)
(C)
MAIN:
211
処理a
call SUB1
SUB1:
213
処理b
call SUB2
SUB2:
215
処理c
210 MAINルーチン
212 SUB1ルーチン
214 SUB2ルーチン
(D)
(E)
(F)
220メモリ上の
スタック領域
220
220
SP 232
222(MAINで
使用するための
スタック領域)
SP 234
224(SUB1で
使用するための
スタック領域)
222
SP 236
226(SUB2で
使用するための
スタック領域)
224
222

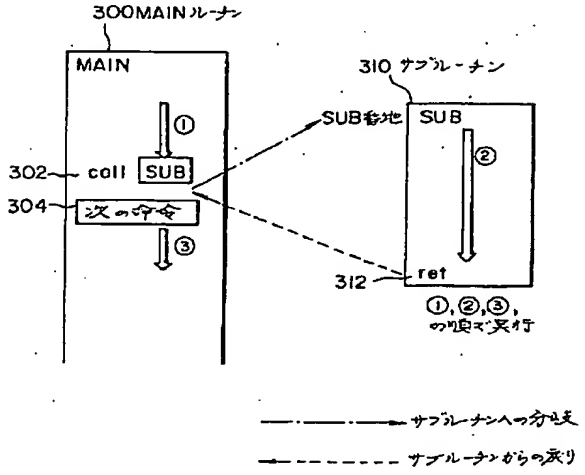
【図１１】



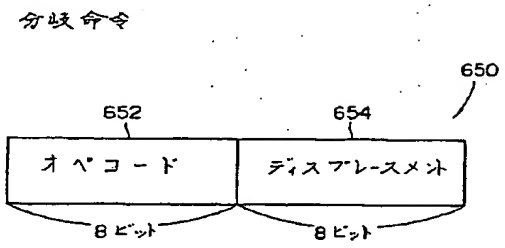
【図１２】



【図１３】

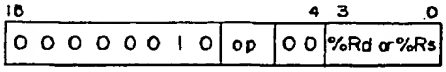


【図１４】



【図１９】

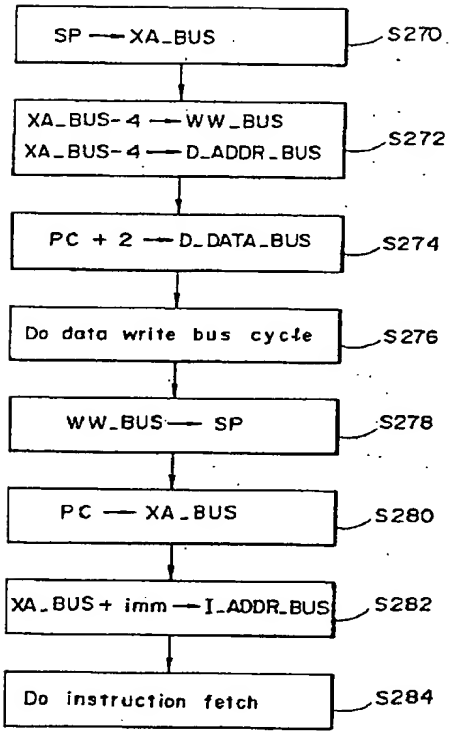
pushn 命令, popn 命令のビットマップ



命令	op
pushn	0 0
popn	0 1

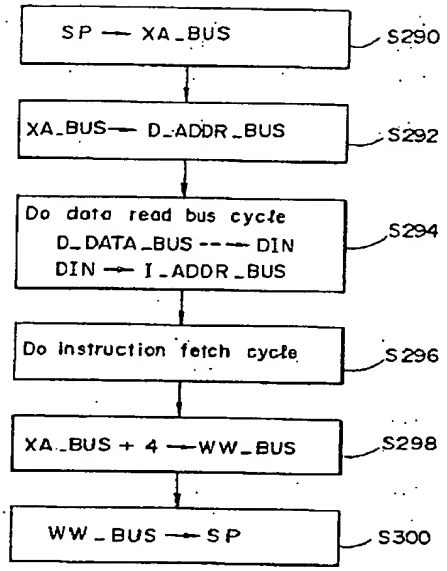
【図15】

call imm



【図16】

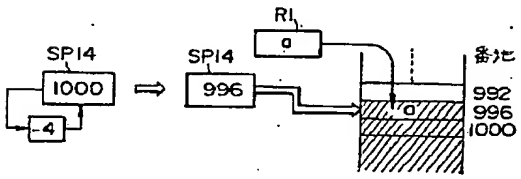
ret



【図17】

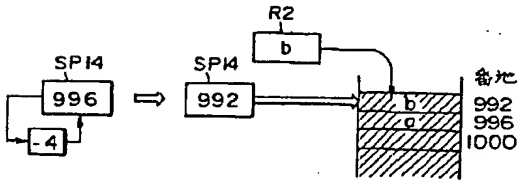
(A)

push R1 実行時の動き

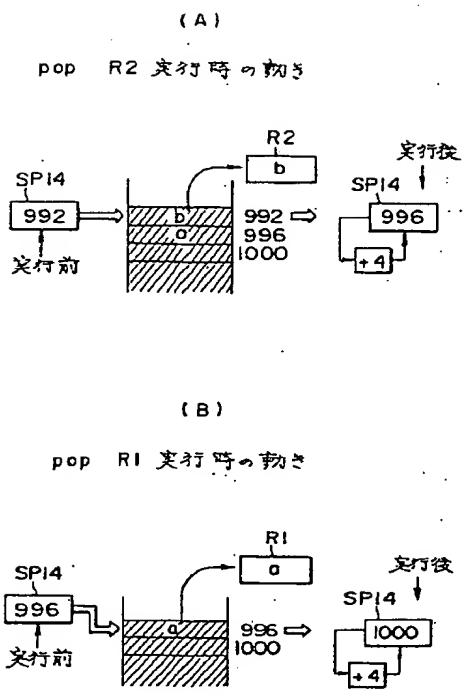


(B)

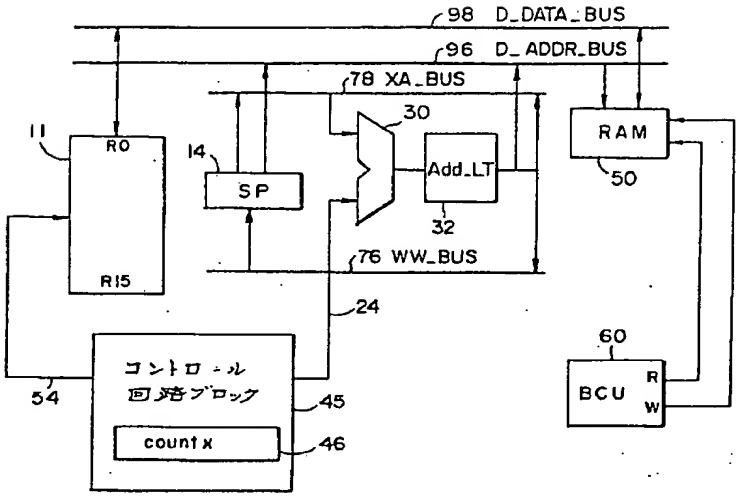
push R2 実行時の動き



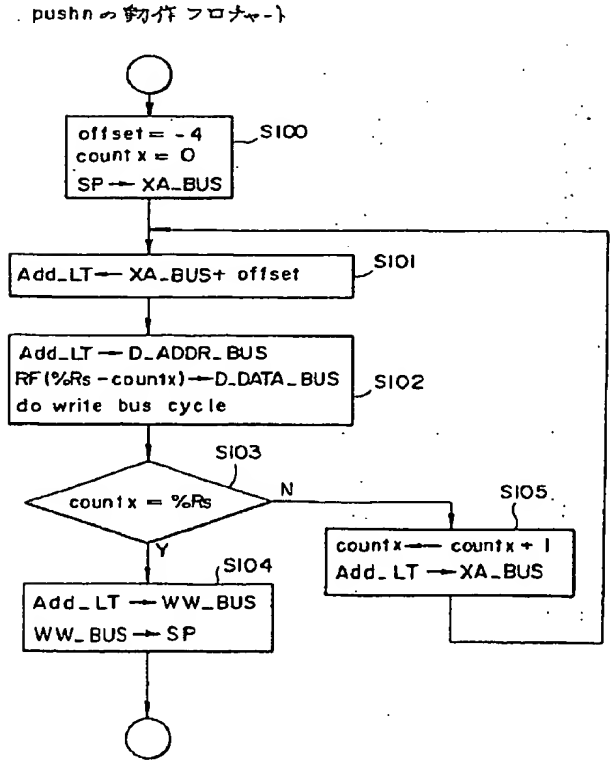
【図１８】


(A)
pop　R2 実行時の動き
SP14
992
実行前
R2
b
a
992
996
1000
実行後
SP14
996
+4
(B)
pop　R1 実行時の動き
SP14
996
実行前
R1
a
996
1000
実行後
SP14
1000
+4

【図２０】

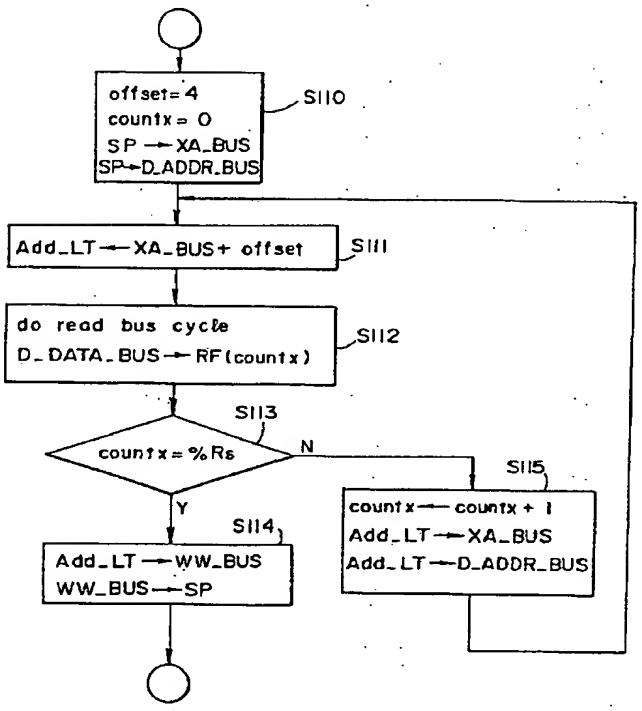

98 D_DATA_BUS
96 D_ADDR_BUS
78 XA_BUS
30
11
R0
R15
14
SP
Add_LT
32
RAM
50
76 WW_BUS
24
60
BCU
R
W
コントロール
回路ブロック
45
54
countx
46

【図２１】

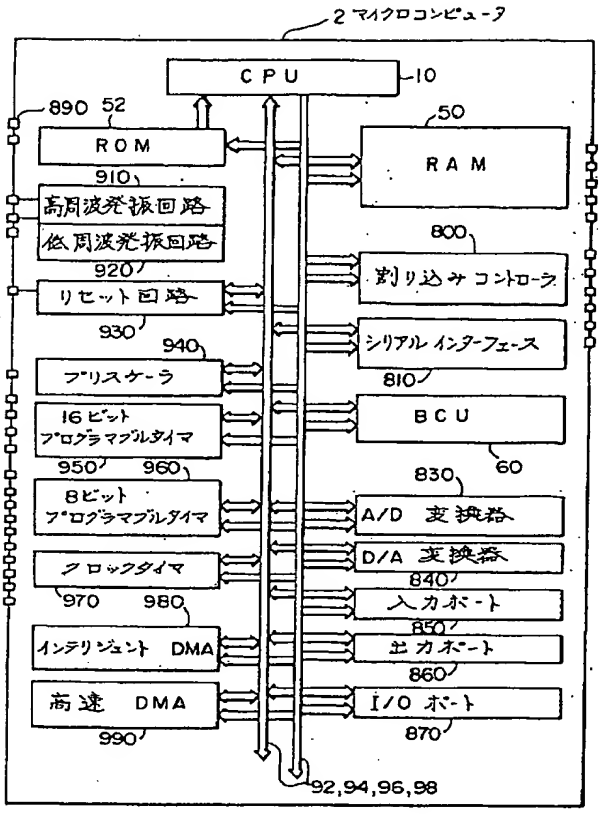

pushn の動作フローチャート
offset = -4
countx = 0
SP → XA_BUS
S100
Add_LT ← XA_BUS+ offset
S101
Add_LT → D_ADDR_BUS
RF(%Rs - countx) → D_DATA_BUS
do write bus cycle
S102
S103
countx = %Rs
N
Y
S105
countx ← countx + 1
Add_LT → XA_BUS
S104
Add_LT → WW_BUS
WW_BUS → SP

【図２２】

popnの動作フローチャート



【図２３】



フロントページの続き

(72)発明者　佐藤　比佐夫
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内